ESD-TR-91-247

Annual Report

Knowledge-Based System Analysis and Control Defense Switched Network Task Areas

30 September 1991

Lincoln Laboratory

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LEXINGTON, MASSACHUSETTS

Prepared for the Department of the Air Force under Contract F19628-90-C-0002.

This report is based on studies performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology. The work was sponsored by the Department of the Air Force under Contract F19628-90-C-0002.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LINCOLN LABORATORY

KNOWLEDGE-BASED SYSTEM ANALYSIS AND CONTROL DEFENSE SWITCHED NETWORK TASK AREAS

ANNUAL REPORT SUBMITTED TO
MR. TOM LAM
DCEC, DRFB
1860 WIEHLE AVENUE
RESTON, VA 22090-5500

H.M. HEGGESTAD
Group 21

1 OCTOBER 1990 – 30 SEPTEMBER 1991

ISSUED 25 JUNE 1992

LEXINGTON MASSACHUSETTS

ABSTRACT

A major activity during FY91 has been the design and implementation of a new Artificial Intelligence (AI) Utility architecture for the Integrated Workstation (IW) used by network management personnel at Defense Information Systems Agency-Europe (DISA-Europe). The IW is the part of the Defense Switched Network (DSN) Integrated Management Support System that is concerned with near-real-time monitoring and control of the network. Development of the new AI Utility architecture has been a joint effort between GTE and Lincoln Laboratory. For the new utility, Lincoln has developed a revised IW Expert System (IWES) that no longer depends on the neural network component of the old architecture for problem detection. In addition, the new IWES has monitors that produce Alert messages when reported values from the network indicate significant departures from normal values recorded in a statistical database. The new architecture also allows expert site personnel to adjust the behavior of the system through new user interfaces.

The IW, as installed in September 1990 and used throughout FY91, was subjected to operational test and evaluation (OT&E) at DISA-Europe in September 1991. The system, including the IWES, met user criteria and was recommended for acceptance as operational. Subsequently the new AI architecture was installed and demonstrated for site personnel.

Substantial work has been done to enhance the value of the Lincoln Call-by-Call Simulator (CCSIM) as a trainer for IW operators. Other enhancements include the completion of the implementation of Common Channel Signaling, speeding up of the LOAD-LEVEL command to support time-of-day traffic changes, and upgrading the graphics interface to operate with Sun OpenWindows Release 2. The CCSIM Top-Level Design Document was issued, and a draft document on the Common Channel Signaling implementation was sent to the Defense Communications Engineering Center (DCEC) for review. Preliminary simulations of the current Pacific DSN were carried out.

A component of DCEC FY91 tasking for Lincoln Laboratory is DRTV-funded expert systems development for Defense Communication System transmission system control. This tasking is referred to as Transmission Monitoring and Control (TRAMCON) alarm integration, as in past reports. It had one major component in FY91, implementation of the TRAMCON Alarm Interpreter (TAI), and one minor component, maintenance and extension of the TRAMCON Event Generator (TEG). TAI has been developed to explore the practicality of using an expert system to aid TRAMCON system operators in diagnosing transmission equipment failures from segment-wide patterns of equipment alarms. A preliminary version of TAI has been implemented and tested, and TEG has been extended to support the testing and demonstration of TAI.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

1. INTRODUCTION AND SUMMARY

A major activity during FY91 has been the design and implementation of a new Artificial Intelligence (AI) Utility architecture for the Integrated Workstation (IW) used by network management personnel at Defense Information Systems Agency-Europe (DISA-Europe). The IW is the part of the Defense Switched Network (DSN) Integrated Management Support System (DIMSS) that is concerned with near-real-time monitoring and control of the network. Development of the new AI Utility architecture has been a joint effort between GTE and Lincoln Laboratory. For the new utility, Lincoln has developed a revised IW Expert System (IWES) that no longer depends on the Neural Network (NN) component of the old architecture for problem detection. In addition, the new IWES has monitors that produce "Alert" messages when reported values from the network indicate significant departures from normal values recorded in a statistical database (STATDB). The new architecture also allows expert site personnel to adjust the behavior of the system through new user interfaces. Section 2 of this report describes the overall architecture, the design of the IWES portion, and the methods used in IWES for problem detection and verification. Other subsections report on the progress and status of work on the STATDB used by IWES and the user interfaces and support tools now available.

The IW, as installed in September 1990 and used throughout FY91, was subjected to operational test and evaluation (OT&E) at DISA-Europe in September 1991. The system, including the IWES, met user criteria and was recommended for acceptance as operational. Subsequently the new AI architecture was installed and demonstrated for site personnel. Section 3 provides background and a discussion of the test results.

Substantial work has been done to enhance the value of the Lincoln Call-by-Call Simulator (CCSIM) as a trainer for IW operators. Other enhancements include the completion of the implementation of Common Channel Signaling, speeding up of the LOAD-LEVEL command to support time-of-day traffic changes, and upgrading the graphics interface to operate with Sun OpenWindows Release 2. The CCSIM Top-Level Design Document was issued, and a draft document on the Common Channel Signaling implementation was sent to the Defense Communications Engineering Center (DCEC) for review. Preliminary simulations of the current Pacific DSN were carried out. Section 4 describes these activities.

A component of DCEC FY91 tasking for Lincoln Laboratory is the DRTV-funded expert systems development for Defense Communications System (DCS) transmission system control. This tasking is referred to as Transmission Monitoring and Control (TRAMCON) alarm integration, as in past reports. It had one major component in FY91, implementation of the TRAMCON Alarm Interpreter (TAI), and one minor component, maintenance and extension of the TRAMCON Event Generator (TEG). TAI has been developed to explore the practicality of using an expert system to aid TRAMCON system operators in diagnosing transmission equipment failures from segment-wide patterns of equipment alarms. A preliminary version of TAI has been implemented and tested, and TEG has been extended to support the testing and demonstration of TAI. Section 5 describes the algorithms used in these programs and the interfaces through which they can be used. It also contains a discussion of desirable future work to enhance their capabilities.

Appendices A and B provide detailed documentation of the IWES system design and the monitors and rules compose the expert system. Appendix C provides details on the process of building a new STATDB for a network.

2. AI ARCHITECTURE DESIGN AND IMPLEMENTATION

The FY90 annual report described the Integrated Workstation (IW) architecture in which our Integrated Workstation Expert System (IWES) was a component. The Artificial Intelligence (AI) portion of that system consisted of GTE's Neural Network (NN) and IWES, and together they were known as the AI Utility. The function of the AI Utility is to detect network problems by analyzing the switch reports and to display them along with suggested actions and controls as well as other useful information. During the early part of FY91, GTE developed programs that allowed the network manager to experiment on-line with anomaly detection algorithms that dealt with a class of problems not recognized by the NN. DCEC requested that these capabilities be integrated with IWES into a new AI Utility architecture. The design and implementation of that new AI architecture as a joint effort with GTE has been a major component of our work in the current fiscal year.

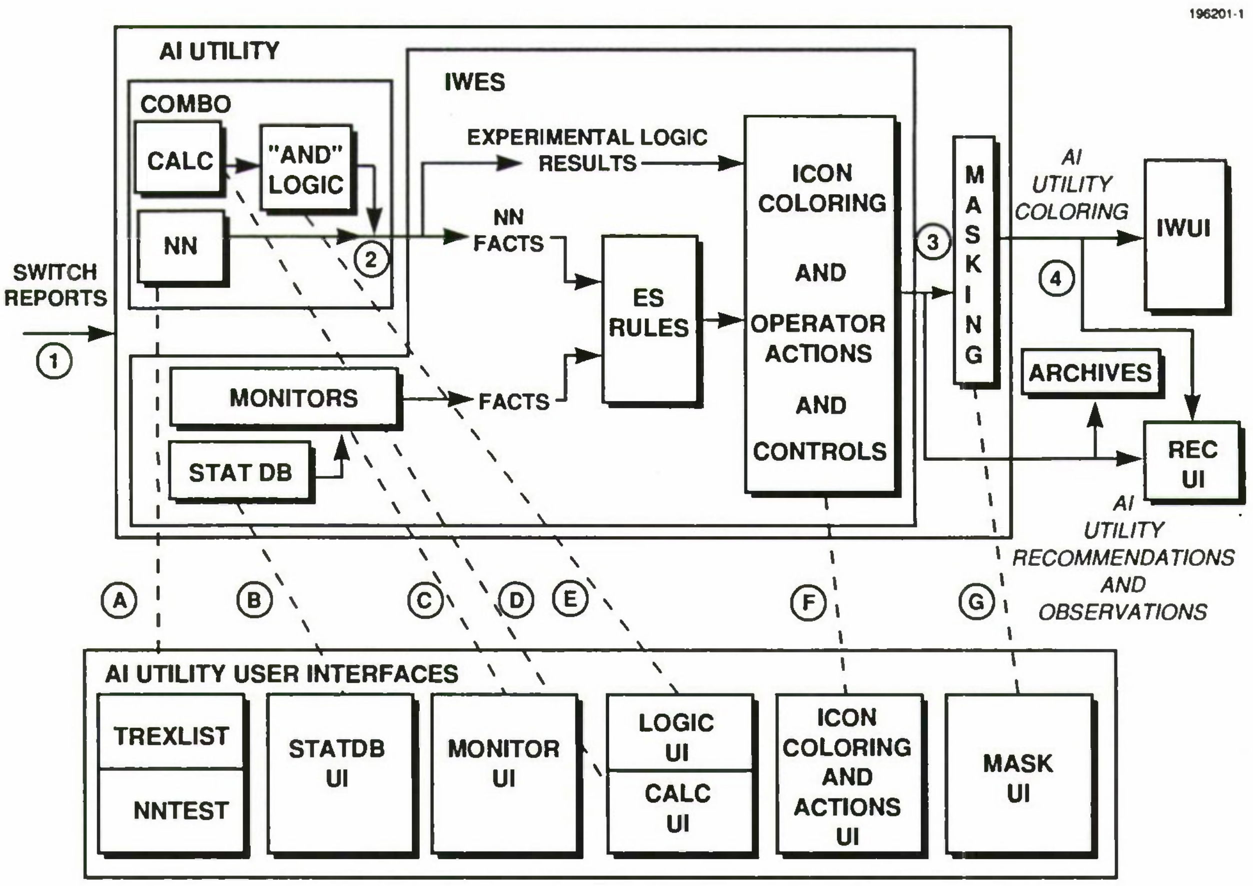
The following subsections describe the overall architecture, the design of the IWES portion, and the methods used in IWES for problem detection and verification. Other subsections report on the progress and status of work on the statistical database (STATDB) used by IWES and the user interfaces and support tools now available. Appendices A, B, and C provide further detail on the design of IWES and the STATDB.

2.1 THE NEW AI ARCHITECTURE

As shown in Figure 1, the AI Utility is composed of three major components called COMBO, IWES, and MASKING. These are made up of modules the functions of which are described below. User interfaces for each module are available for a qualified network management expert to adjust and control the behavior of the module in the overall system. Results are displayed to the network management expert and other IW operators via the Integrated Workstation User Interface (IWUI) and the Recommendations User Interface (REC UI) processes. AI Utility recommendations and observations are archived over time.

As displayed in Figure 1, switch reports enter the AI Utility on the left. They are processed by the NN and CALC modules of the COMBO component and by the expert system MONITORS module. The AND logic makes use of calculations available from the CALC program and simple logic rules from the LOGIC UI to detect experimental anomalies. COMBO combines the anomalies detected by the NN with the experimental results and writes them to a file.

The IWES MONITORS module processes switch reports and uses fixed thresholds and normal values accessed from the STATDB to detect abnormal switch report register values and quantities calculated from such report values. These abnormalities are asserted as FACTS into the IWES knowledge base where rules detect network problem events. IWES makes use of an expert system shell developed at the NASA/Johnson Space Center called CLIPS (C Language Integrated Production System). As shown in the figure, a statistical database user interface (STATDB UI) would allow a network management expert to create new STATDBs. Actually, creation of a new database is a lengthy, off-line procedure, and there is no STATDB UI module as such to support this function. However, programs are available to support the creation and modification of the STATDB (see Appendix C), and, because the module was shown



DATA INTERFACES

1. Raw data is read in with calls to the IW database.
2. Neural network and experimental logic outputs are written to a file. This file is rewritten each time step.
3. Problem descriptions, icon coloring, observations, and recommended actions and controls are written to a file that is archived for each time period.
4. Unmasked problems and associated colors are written to a file. This file is rewritten each time step.

CONFIGURATION INTERFACES

A. TREXLIST and NNTEST are used to train the neural network.
B. STATDB UI is used to update the statistical database.
C. MONITOR UI is used to adjust threshold values for monitors.
D. CALC UI generates equation coefficients used to produce CALC parameters.
E. LOGIC UI generates a set of "AND"-type rules.
F. ICON COLORING AND ACTIONS UI generates a file containing a lookup table for colors and recommended actions.
G. MASK UI feeds the MASKING program elements to be masked.

Figure 1. AI Utility architecture.

on earlier drawings of the AI architecture, it is retained in Figure 1. The figure also shows a MONITOR UI allowing a network management expert to modify thresholds and change standard deviations when necessary. This on-line capability is present in the IW, but the MONITOR UI is accessed via an IWES UI that allows control of other aspects of IWES operation. The IWES UI is not shown in the figure because it is an addition to the agreed-upon architecture and is not essential to the AI Utility functions. (See Section 2.6 for more on the IWES UI.)

IWES also reads in the output file generated by COMBO and asserts NN results as NN FACTS into the IWES knowledge base. NN-related CLIPS rules confirm NN anomalies using CLIPS facts asserted by the monitors. The output of the expert system rules portion of IWES is a list of network problem events for the time period.

The ICON COLORING, OPERATOR ACTIONS' and CONTROLS module receives both network problem events and experimental logic results and writes a file containing problem descriptions, icon coloring, recommended operator actions and controls, and observations. This file is archived for logging purposes and is also read by the MASKING program and the REC UI. The ICON COLORING and ACTIONS UI allows the network manager to modify icon (switch or trunk group) coloring and recommended actions to be associated with any problem type that the AI Utility might detect.

MASKING accesses a file, created by the network manager via MASK UI, listing problems to be masked and not displayed by the IWUI. It writes a list of anomalies that are not masked and their associated icon colors to yet another file that is read by the IWUI. Any masks that fail to mask a problem in a particular report interval are sent to a user display indicating that a previously masked problem did not occur in the current time period. This facility allows the operator to easily detect that a condition to be masked is no longer present.

The IWUI reads in the list of anomalies and icon colors and updates the display appropriately. When a switch or trunk group icon is moused on the IWUI, a message is sent to the REC UI, and the REC UI displays the AI Utility recommendations and observations related to the icon.

The role of the Experimental Logic is seen as giving a network management expert a way to experiment with new anomalies that he or she would like to see detected by the IWES monitors. When the network management expert has solidified recognition of the anomaly, rules can be added to the expert system to detect this anomaly and more specific instances of the problem. More complex rules that look at statistically normal values and calculations for the time of day, information from neighboring switches and trunk groups, anomalies over time, etc. can be introduced to increase the reliability of detection. COMBO limits the recognition to anomalies that are detectable directly from the values returned in a single switch report.

Implementation of the new AI architecture has been a joint effort with GTE. Lincoln Laboratory has had responsibility only for IWES and its related interfaces. The remaining components of the system have been developed by GTE and are not discussed further in this report.

2.2 THE FY91 IWES

The FY90 IWES processed raw data from switch reports, STATDB values, and NN results. In the FY90 AI architecture, problems were detected only by the NN and were validated by IWES. The problem detection capabilities of the predecessor network management expert system were turned off in the FY90 AI architecture. Validation involved assuring that the IWES monitors detected corresponding interesting features, that were abnormal for the time of day, in the raw data.

During the September 1990 DIMSS testing, it was observed that the expert system's monitors, which were running in preparation for validating problems that might be detected by the NN were, in fact, making observations about raw switch and trunk group data that would have allowed IWES to detect some problems before the NN detected them. During FY91, Lincoln Laboratory was asked to reinstate the original expert system monitors and rules, along with adding new STATDB monitors and related rules. In addition, Lincoln Laboratory was tasked to continue to add monitors and rules resulting from ongoing knowledge engineering. IWES was also expected to continue to deal with verification of anomaly detections made by the NN.

Figure 2 shows the current IWES structure. When IWES is initialized, network representation and STATDB values are read in and stored in C language structures. During each polling period, switch reports and NN anomalies are received, processed, and stored in other C structures. These structures are scanned by monitors (implemented in C) to identify data outside the statistical norm for the time of day, NN anomalies, and other interesting features and useful information. The outputs of the monitors are asserted into CLIPS as FACTS representing the abstract state of the network. Problems are identified and validated by a module composed of CLIPS rules that looks for patterns of symptoms in the abstract state of the network.

Identification of problems by the expert system is accomplished by rules that combine specific interesting features in the raw data that were abnormal for the time of day. Although the NN is trained to detect very few problems, validation of a problem announced by the NN is still in place and is accomplished by ensuring that IWES monitors detected corresponding interesting features in the raw data. All problems identified, both validated and unvalidated, are confirmed over time by a confirmation module. A planning module assigns actions and controls to improve or correct the situation. An observation module observes the effects of the applied controls over time and recommends removal of the controls when the problem no longer exists. Each module adds text observations to a C structure that contains the history of current problems for each element of the network.

2.3 NETWORK PROBLEM EVENT DETECTION

IWES can be run with or without input from the STATDB monitors, the NN, or the expert system monitors. IWES invocation options allow the user to indicate the types of input he or she would like to use and the associated rules that will be loaded. Rules dealing with expert system network problem events are independent from NN rules unless they both detect a problem. When this happens, the rules for validating the detected NN anomaly are activated and the rules for detecting the corresponding expert system network problem event are deactivated. The result is a validated NN anomaly. When IWES is run without NN input and with IWES monitors, the rules relating to NN anomalies are not loaded and problems are detected by expert system network problem event rules only.

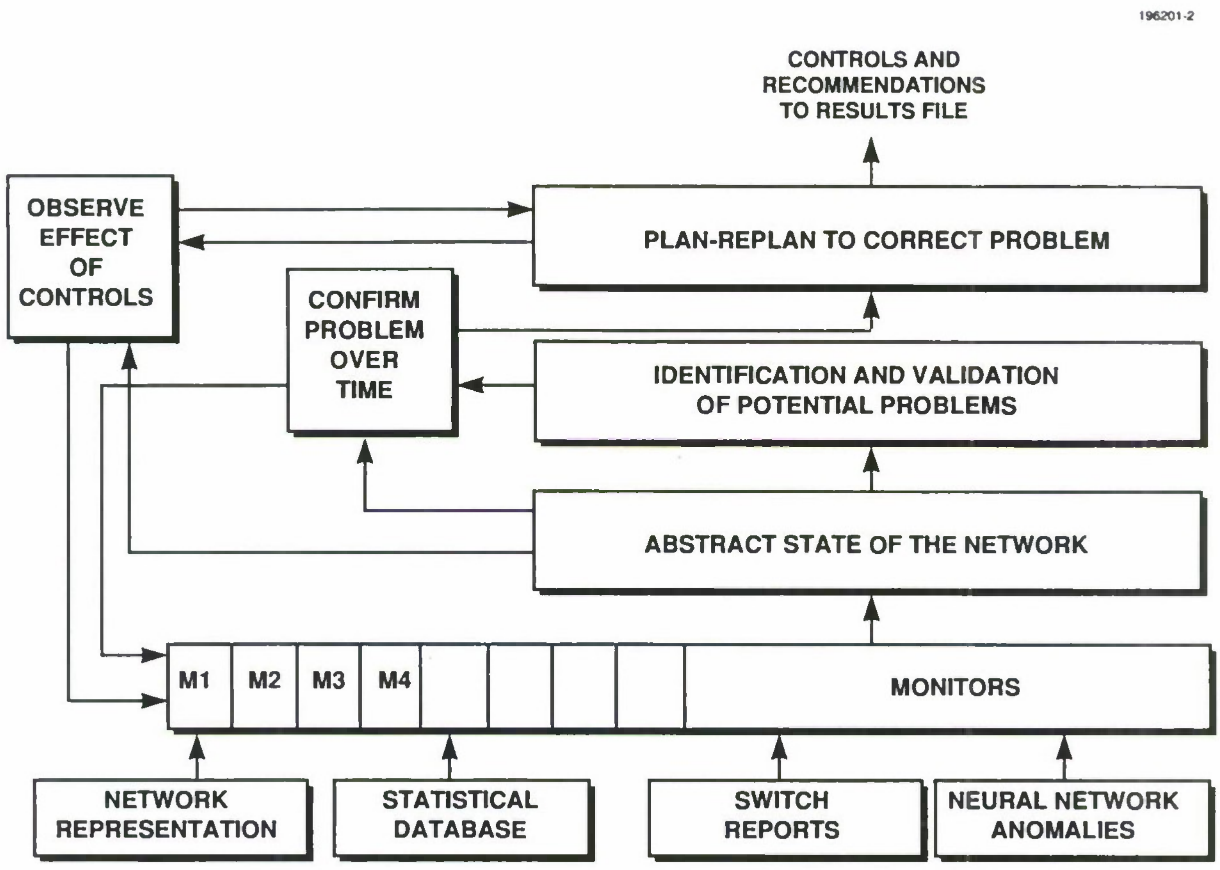

196201-2
CONTROLS AND RECOMMENDATIONS TO RESULTS FILE
OBSERVE EFFECT OF CONTROLS
PLAN-REPLAN TO CORRECT PROBLEM
CONFIRM PROBLEM OVER TIME
IDENTIFICATION AND VALIDATION OF POTENTIAL PROBLEMS
ABSTRACT STATE OF THE NETWORK
M1
M2
M3
M4
MONITORS
NETWORK REPRESENTATION
STATISTICAL DATABASE
SWITCH REPORTS
NEURAL NETWORK ANOMALIES

Figure 2. IWES structure.

There are two types of rules for identifying and validating problems detected by IWES and its STATDB monitors. One type is a set of general rules that recognize, monitor over time, relate and report network problem events; the other type is a set of specific rules that match combinations of monitor results to create problem facts for each problem to be used by the general rules. Additional details are found in Appendix B (IWES Monitor and Rule Descriptions) in Section B.2.1 (General Rules for Detection of Network Problem Events).

The problem fact created by a specific rule is written in a format that can be matched in a general rule to create a network problem event. Once a network problem event is created, additional general rules update it over time and detect the removal of the problem. CLIPS structures called "deftemplates," which are similar to frames and define a group of related fields in a pattern, are used to define network problem events. Each switch problem event has fields for storing the problem name, switch, number of times the problem has occurred, the present time, whether the problem should be reported, an action to be taken, etc. Trunk group problem events have similar fields along with a trunk group name and a trunk group destination.

Some problems can be overridden by more specific problems. For example, if there are excess trunks out of service (more than 50 percent) then the operator is not interested in knowing that there are significant trunks out of service (more than 25 percent). As another example, if there is severe multi-frequency (MF) receiver overflow the operator does not need to know what actions to take for less significant problems such as MF receiver overflow and high MF receiver overflow. The operator should not be overloaded; he or she should be informed about the most severe problems as well as problems for which actions should be taken. Other problems that are less significant are listed as observations, as suppressed problems, or sometimes as secondary problems. "Deftemplates" are used to define types of problems. They have problem names, element types (switch or trunk group) and a list of other problems that are to be overridden when the problem occurs. A general rule matches problem events and problem types and suppresses any other problem events at the switch or on the trunk group that are listed to be overridden when the original problem event occurs. Problem types are asserted from C code and eventually could be read in from a file that could be modified by a network manager directly or through a user interface.

A final set of general rules calls C routines to store unsuppressed problems in the current problem array and all problems in the observations array.

Most of the rules for diagnosing the problems recognized by IWES were developed by the European network manager, Mr. Thomas, using the Experimental Logic system, and translated into CLIPS rules by Lincoln Laboratory. The Experimental Logic program, which was designed and implemented by GTE, has proved to be an excellent knowledge engineering tool, not only because it captures the actual rules that the network management expert uses to diagnose problems, but it also allows the expert to increase his/her knowledge and refine his/her rules through experimentation. It also gives Lincoln Laboratory a basis to discuss more complex network problem events with the network management expert that the Experimental Logic program is unable to capture. Some of these more complex network problem events involve deviations from statistically normal data, information from neighboring switches and trunk groups, or recognition over time. Examples might be abnormally high call condense block (CCB) counts and trunks being permanently seized with unsupporting peg counts.

In the present version of IWES, specific rules and associated C code have been added to recognize the following list of basic problem events. All except for those marked with asterisks were defined by the expert and developed using the Experimental Logic system.

Switch Problem Events
- Miscellaneous others
- Serious switch trouble
- Permanent signals
- Central processor traps
- Call condense block overflows
- Match Initialization
- Trunk group hit
- MF receiver overload
- High MF receiver overload

Severe MF receiver overload
Digitone receiver overload
High Digitone receiver overload
Severe Digitone receiver overload
Span Diagnostics
* Abnormally high call condense block count

Trunk Group Problem Events
Glare
Outfailures
Infailures
Excessive trunks out of service
Significant trunks out of service
Significant problem - Glare
* One or more trunks permanently held up with unsupporting peg counts
Restricted trunks

2.4 NN ANOMALY VERIFICATION

The NN-related rules described in the FY90 annual report have also been restructured. There are now 16 general rules, eight for switch and eight for trunk group problems, dealing with confirming and monitoring NN anomalies over time. These rules match on an NN fact and an IWES fact for confirmation of a problem and only an NN fact for an unconfirmed fact. Other functions of these rules include watching the problem over time and detecting the removal of the NN or IWES fact. For each NN anomaly there should be a rule to create an IWES confirmation fact. At this point, some of the NN anomalies are automatically confirmed because of lack of information about them. This modification has made it quite a bit easier to process a new NN anomaly. Some new C code is needed, but only one rule has to be added to the knowledge base for each new NN anomaly.

In the present version of IWES, specific rules and associated C code exist to recognize and verify the following NN outputs:

Switch Anomalies
Outage remote
Degraded remote
Degraded reporting

Trunk Group Anomalies
Trunk signaling problems
Tropofade
Permanent seizure

The above list consists of all the NN anomalies for which it appears that NN training has been carried out during the latter part of FY91. Many of the original NN training cases have either been removed from the training list and/or are now recognized by experimental logic rather than the NN.

2.5 IWES STATDB

The major effort for the STATDB this year was turning last year's demonstration, proof-of-concept system into an integral part of the operational IWES system. IWES now has STATDB monitors that generate Alert messages for the operator when values are observed that fall outside user-specifiable limits. In addition, IWES recommendations include supporting information derived from the STATDB.

The STATDB now has a full set of stat parameters for every reporting switch and trunk group. There are eight stats for each switch and eight stats for each trunk group. They are stored as hourly averages, and interpolation is used to derive appropriate values for any minute of the day. Figure 3 shows an example of the fit between interpolated average traffic values from the database and actual 15-min reports for a typical day at a large switch. Lincoln Laboratory believes that linear interpolation of hourly averages provides sufficiently accurate estimates at times of the day when the traffic is changing rapidly from period to period; its use results in significant space savings in the database.

IWES now provides for automatic weekend/holiday and daylight savings time adjustments with operator overrides. Switch reports are always timestamped with Greenwich Mean Time (GMT). However, traffic patterns are dependent on work hours that change when daylight savings time changes and on weekends. Using system time calls and the STATDB date information, IWES determines if database lookups must be offset by 60 min to compensate for daylight savings time effects.

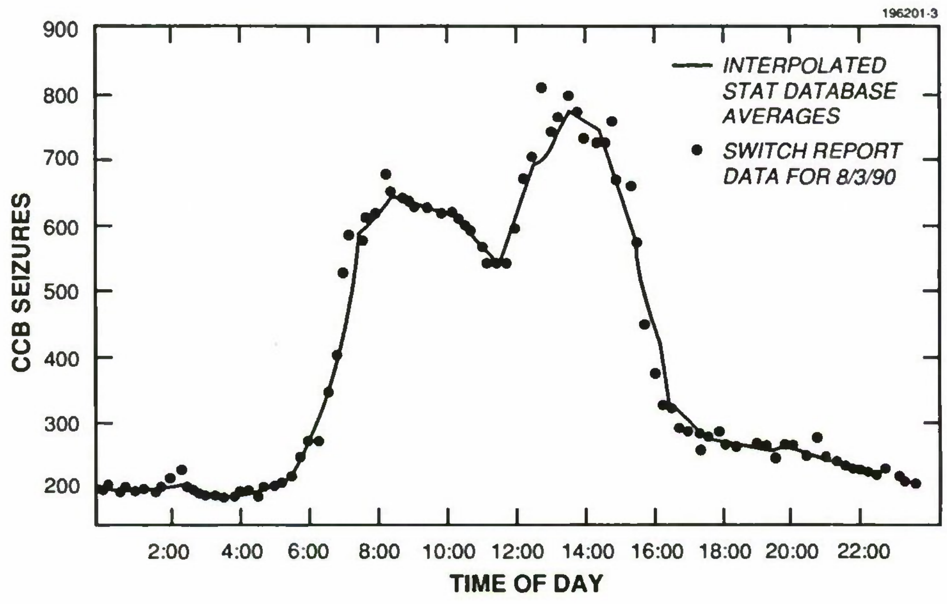


Figure 3. Interpolated database traffic data example.

A new smoothing algorithm is employed in IWES to deal with noisy 5-min switch report data. The algorithm maintains a linked list of switch reports for every switch. The list varies in length depending upon the traffic intensity at each switch and time of day. The algorithm ensures that enough calls (currently 35) and enough time (currently 15 min) are considered when computing values for comparison with the database. The parameters of 35 calls and 15 min as well as a threshold factor of plus or minus two and one-half standard deviations are derived from analysis of and testing on significant quantities of archived data. Lincoln Laboratory believes that smoothing algorithm provides the operator and IWES monitors with accurate switch stats regardless of the time of day or the size of the switch.

In order to obtain accurate normal values and ranges for the database, it is important to identify and remove the abnormal values due to data link errors and network anomalies. To support this goal, the STATDB compiler was enhanced with a second pass that computes new averages and standard deviations after eliminating input values outside three standard deviations from the tentative average computed in the first pass.

To avoid false alarms during start-up and when processing archived data out-of-sequence, the data accumulators used in the smoothing algorithm are now initialized with normal data from the STATDB. Immediate threshold checks can now be made without having to run through a number of cycles. When looking at archived data, the network analyst is now able to jump to different report times and the stat monitors will reinitialize appropriately.

Switch stat monitors now detect any switch report stat value that is outside its expected range for the specific time of day and switch. The monitors generate Alert messages for the operator and log. All eight switch stats are monitored: MF and Digitone calls, MF and Digitone receiver usage, MF and Digitone receiver holding time, CCB seizures, and the ratio of calls to CCB seizures. MF calls are interswitch calls. Digitone calls come from local telephones and PBXs (Private Branch Exchanges).

Some new switch fields that became available in the switch reports late last year were analyzed at the request of the European network analyst. A number of the switch error codes were found to be highly dependent on traffic load and thus excellent candidates for the STATDB. They have not yet been incorporated into the STATDB.

The STATDB also provides collateral information for switch and trunk group problems that appears in the recommendations generated by IWES. In addition to the switch stats described above, the STATDB provides for each trunk group: incoming calls, outgoing calls, attempts, overflows, usage, and holding time.

The status line in the IWES log window (see Section 2.6) now includes two network traffic monitors that provide the operator with an overview of network traffic and a verification of the STATDB. The %NORMAL_TRAFFIC TODAY is a running comparison (percentage) of the total number of CCBs used to the expected number. Because a CCB must be seized for a switch to handle a call, CCB activity provides a useful estimator of network traffic. The expected running total is recomputed each report period by adding in the corresponding STATDB value for each reporting switch. The %NORMAL_TRAFFIC NOW is a comparison of the total CCBs in the current timestamp with the corresponding total from the STATDB for that time. With these stats, the network analyst can monitor long-term network traffic changes and detect and monitor temporary traffic fluctuations. If the network stats continually show a difference between expected and actual, then the analyst should compile a new STATDB.

In order both to provide better support and to lessen the need for Lincoln Laboratory on-site support, a number of utilities were implemented or enhanced. They allow a UNIX programmer, without detailed knowledge of the STATDB, to compile a new database using the standard IW network configuration table and switch data. More information on the procedure for generating a new database can be found in Appendix C.

Documentation on how to build, install, and use the STATDB was provided to the sponsor and support contractor in Europe. Sun XWindows Version 2 and the STATDB were installed and demonstrated at DCEC.

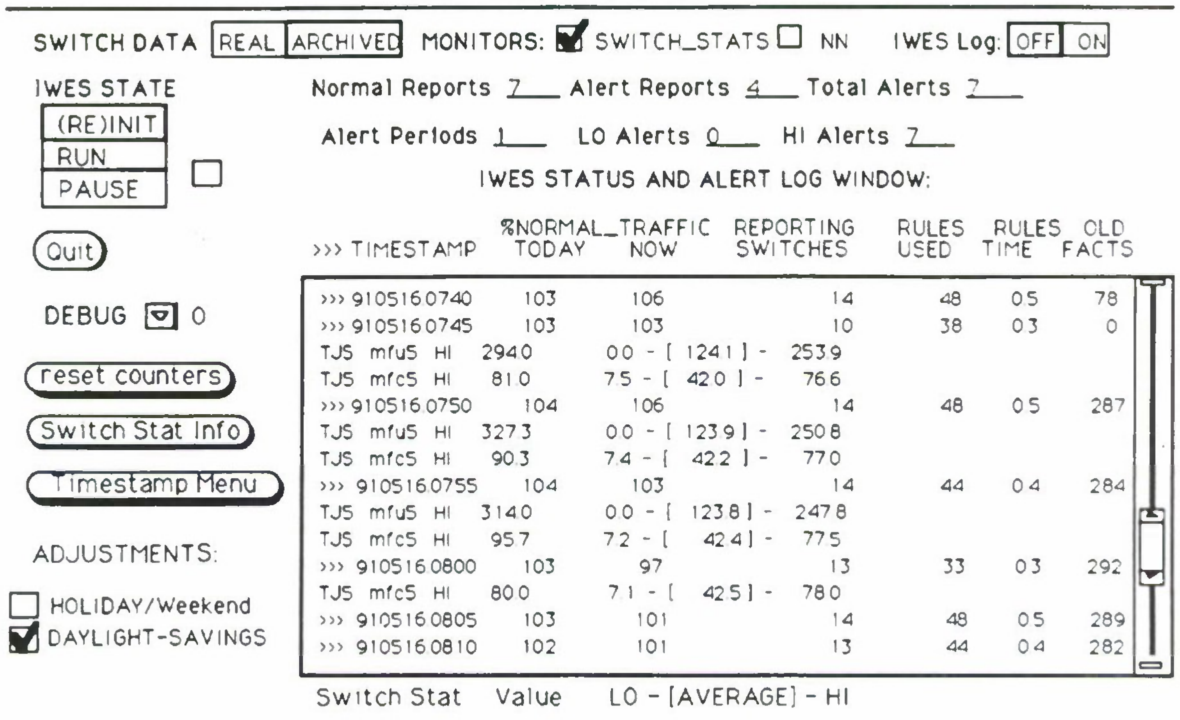
To support the validation of the STATDB, a set of graphs was supplied to DCEC. The graphs plotted input switch data points against the resultant statistical file values. Scatter points (no connecting lines) represented the actual values from the sample switch reports used to compile the database. There were approximately 80 values per hour representing the seven sample days with 12 reports per hour. The STATDB mean, mean plus one standard deviation, and maximum were plotted as lines. All were computed after filtering the outliers exceeding three standard deviations from the mean during the first pass of the database compiler. The figures clearly showed that there were a number of outliers in the sample data representing either data link errors or network anomalies. Lincoln Laboratory believes that the figures also showed that the database values were a reasonable representation for the real data.

2.6 IWES USER INTERFACES

Three IWES user interfaces are now part of the IW. One, the IWES UI, was initially developed as a front end for demonstration and testing of the STATDB. It has been modified to support general user interaction with the IWES system. The second, accessed via the Switch Stat Info button on the IWES UI, provides more detailed information about the Alerts detected by the STATDB monitors, and allows the operator to adjust parameters in those monitors. It can be identified with the MONITOR UI shown in the AI Utility architecture (Figure 1). The third is the REC UI in Figure 1, which displays the recommendations outputs of IWES. In the old IW architecture, the function of this latter UI was carried out in GTE's IWUI. In the new architecture, the REC UI was separated from the old IWUI so that the staff at Lincoln Laboratory could change the format of the recommendations presentation without requiring GTE to change any software.

Figure 4 shows a representation of the IWES UI display. All user-settable controls and additional menu buttons are along the top and left borders of the window. The remaining items are IWES outputs: running counters of normal (no stats out of range) and abnormal report intervals, and a scrollable stat log panel with times and data about each stat detected out of normal range. Each time IWES completes the analysis of all switch data for a report period, it updates the counters and the log panel.

The figure shows the TEST version of the IWES UI. This version allows expert users to run tests and demonstrations using archived data. With the "Timestamp" submenu the user selects a sequence of one or more sets of timestamped switch reports from the IW archived data directory and then uses "RUN" to process the data. The "PAUSE" and "(RE)INIT" commands allow the user to suspend or terminate a test.



Normal Reports 7 Alert Reports 4 Total Alerts 7

Alert Periods 1 LO Alerts 0 HI Alerts 7

IWES STATUS AND ALERT LOG WINDOW:

```
                    %NORMAL_TRAFFIC   REPORTING     RULES   RULES   OLD
>>> TIMESTAMP        TODAY     NOW     SWITCHES      USED    TIME   FACTS
>>> 910516.0740       103      106           14       48      0.5     78
>>> 910516.0745       103      103           10       38      0.3      0
TJS  mfu5  HI   294.0    0.0 - [ 124.1 ] -  253.9
TJS  mfc5  HI    81.0    7.5 - [  42.0 ] -   76.6
>>> 910516.0750       104      106           14       48      0.5    287
TJS  mfu5  HI   327.3    0.0 - [ 123.9 ] -  250.8
TJS  mfc5  HI    90.3    7.4 - [  42.2 ] -   77.0
>>> 910516.0755       104      103           14       44      0.4    284
TJS  mfu5  HI   314.0    0.0 - [ 123.8 ] -  247.8
TJS  mfc5  HI    95.7    7.2 - [  42.4 ] -   77.5
>>> 910516.0800       103       97           13       33      0.3    292
TJS  mfc5  HI    80.0    7.1 - [  42.5 ] -   78.0
>>> 910516.0805       103      101           14       48      0.5    289
>>> 910516.0810       102      101           13       44      0.4    282
Switch Stat     Value     LO - [AVERAGE] - HI
```

Figure 4. Sample IWES UI display.

The "on-line" IWES UI (not shown) has a subset of the TEST version user controls, but the same IWES outputs. It provides the operator with the controls, counters, and displays that are needed when processing real data with the online IW system.

The outputs displayed in Figure 4 resulted from running a series of 11 successive switch reports from 16 May 1991 archived data. The operator log for that date showed that the network analyst suspected that the TJS switch was experiencing a major glare problem on one of its trunk groups at 7:55 a.m.

The user interface (UI) counters indicate that the STATDB monitors judged seven of the 11 periods to be normal periods and found that the remaining four showed a total of seven "HI" Stat Alerts. Because the Alert Reports were successive, i.e, those for 7:50, 7:55, 8:00, and 8:05, the UI indicates that there was only one "Alert Period." This information is intended to help the operator determine that the Alerts were possibly related to a single problem rather than a number of separate problems.

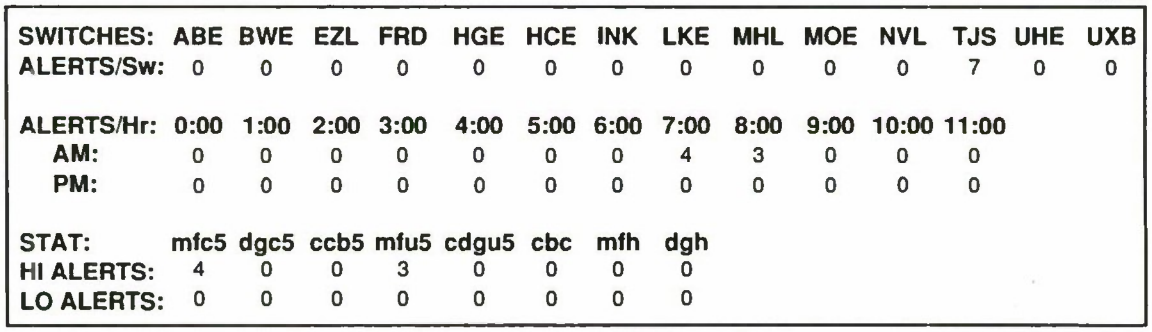
Just above the scrollable log panel are the labels for the IWES status line that is displayed each time IWES completes its analysis of new switch data. The status line, identified by the ">>>" prefix, shows comparisons of actual network traffic with expected traffic, the number of switches reporting, and information about the expert system rules used. The output indicates that traffic for the day and for each report period was between 101 percent and 106 percent of the normal traffic for that time and that at 7:45 a.m. switch data was received from only 10 switches of the possible 14. Normally about 300 facts, asserted in early analysis, become old and are deleted from the IWES facts base each new period. The first two status lines show smaller values because of start-up effects. The rule processing time and the number of rule firings are intended to help us monitor the performance of the IWES inference engine.

When an abnormal switch value is detected, a stat Alert message is displayed in the log window. Its labelling is shown just below the log panel. The example shows that the stat monitors detected abnormal MF receiver stats at TJS from 7:50 till 8:05. The stat Alert message shows the switch, problem type, and actual value vs normal range. For example, at 8:05, the MF call count per 5 min (mfc5) at TJS was 80.0, which is outside of the range considered normal (7.1 to 78.0) for that switch at that time of day.

By selecting the "Switch Stat Info" submenu button, a new window comes up with the detailed breakdown of the stat Alerts shown in Figure 5. The data shows that all seven Alerts occurred at switch TJS, that the Alerts occurred during the 7:00 and 8:00 a.m. hours, and that the problems were all MF related.

The window in which the data of Figure 5 is displayed has buttons and data fields (not shown) that allow the operator to modify the standard deviation range factors used in the STATDB monitor thresholding; the operator can also adjust the number of calls and time parameters of the smoothing algorithm that is applied to the switch report data. This window is the MONITOR UI window shown in the new AI architecture representation (Figure 1).

196201-5


```
SWITCHES:   ABE  BWE  EZL  FRD  HGE   HCE  INK  LKE  MHL  MOE  NVL   TJS   UHE  UXB
ALERTS/Sw:   0    0    0    0    0     0    0    0    0    0    0     7     0    0

ALERTS/Hr:  0:00 1:00 2:00 3:00 4:00  5:00 6:00 7:00 8:00 9:00 10:00 11:00
   AM:       0    0    0    0    0     0    0    4    3    0    0     0
   PM:       0    0    0    0    0     0    0    0    0    0    0     0

STAT:       mfc5 dgc5 ccb5 mfu5 cdgu5 cbc  mfh  dgh
HI ALERTS:   4    0    0    3    0     0    0    0
LO ALERTS:   0    0    0    0    0     0    0    0
```

Figure 5. Stat Alert details — part of MONITOR UI window.

3. OPERATIONAL TEST AND EVALUATION

3.1 SUMMARY

The Integrated Workstation (IW), as installed in September 1990 and used throughout FY91, was subjected to Operational Test and Evaluation (OT&E) at DISA-Europe in September 1991. The system, including the Integrated Workstation Expert System (IWES), met user criteria and was recommended for acceptance as operational. Subsequently, the new Artificial Intelligence (AI) architecture (described in Section 2.1 of this report) was installed and demonstrated for site personnel.

3.2 BACKGROUND

As described in the preceding two annual reports, demonstration models of the Expert System were installed at DISA-Europe in September 1989 and September 1990. The earlier one (the Network Management Expert System or NMES) ran in conjunction with the PC-based Network Management Support System (NMSS). The 1990 system was the IWES, delivered as a component of the IW that was a joint effort of DISA-Europe, DCEC, GTE, and Lincoln. The IW in turn is a major subsystem of the DIMSS. This report focuses primarily on the Expert System components of these systems in that the broader descriptions have been covered in numerous DISA reports and briefings.

It was determined by DISA that an OT&E of DIMSS would be conducted at DISA-Europe from 16 to 20 September 1991. The target system, including the IWES, was delivered in September 1990.

It was determined by DCEC that an upgrade of DIMSS would be installed at DISA-Europe during the week following the OT&E, specifically featuring installation and testing of the new AI architecture developed during FY91.

3.3 TEST RESULTS

DISA and DCEC have briefed and reported the outcome of the DIMSS OT&E; this report emphasizes only the IWES-related aspects of it.

On Monday, 23 September, DISA/DEO described the results of the OT&E in a briefing to the Commander, DISA-Europe. It was stated that DIMSS satisfactorily met operational performance criteria that had been established by DISA-Europe personnel, and it was recommended by all concerned that DISA-Europe take steps to accept DIMSS as an operational system. This acceptance explicitly included the IWES as a component of DIMSS. The recommended vehicle for the acceptance was a Commissioning Agreement that was currently under preparation for the signatures of the cognizant DISA managers. The Commissioning Agreement spelled out approximately 13 items that were mutually agreed upon by DISA-Europe and DCEC as reasonably requiring completion and/or enhancement. As none of these action items affected the IWES, they are not discussed here.

During the week after the OT&E, a group of high-level DISA visitors toured the ACOC, and DIMSS was demonstrated to them by an enlisted systems control operator. This individual, a Navy Chief Petty Officer, was emphatic about the effectiveness and the advantages of the IW. In particular, he praised the problem descriptions, explanations, and recommendations for action that the IWES contributes to the displays. He said he had been very skeptical of the promised benefits of the system at first, but had been won over. He cited examples where the IWES displays had enabled him to recognize a problem condition and take action to correct it, before the users had even begun to notice the effects of the problem.

Also during the week after the OT&E, the new AI architecture was installed and demonstrated to site personnel. Its functions were individually verified for the DCEC program manager (Mr. Lam). A policy was developed in which the new AI architecture can be readily brought up during normal working hours for the on-site GTE network management experts to use and work with it, but after hours the old system will be activated, so as to avoid confusing the military operators. When the time is right and confidence in the new system has been achieved, the military operators can be trained in the new functions and a formal cut-over to the new system can be made.

4. CCSIM STATUS

4.1 IW TRAINER

Substantial work has been done to enhance the value of the Call-by-Call Simulator (CCSIM) as an Integrated Workstation (IW) TRAINER. Parameters are now available to allow the user to specify the level of occurrence of six switch report fields that cannot be directly simulated. For example, the occurrence of glare on a trunk is not simulated by the normal call processing functions in CCSIM. However, the user can specify the value of a variable in the *<net>.inval* file, which will cause glare to be reported for each trunk group as a percentage of the calls on the trunk group. The percentage reported will be an exponentially distributed random value with a mean equal to the variable.

To use CCSIM as an IW TRAINER the training supervisor runs CCSIM through a scenario of interest for training purposes with the `toiwdb` flag set in the *<net>.inval* file. This flag causes the generation of switch reports for the IW Database (IWDB) using naming conventions and directory structures identical to those used at the European ACOC. The naming conventions use the date and time at which the switch report was generated. CCSIM now allows two new optional command line arguments so that the user can provide arbitrary dates and times for the reports to be generated. The date is specified as MM/DD/YY and the time as HHMMhr in standard military form. If CCSIM detects an error in either format, a message is printed and the simulation terminates. These arguments are now accepted by *runsys*, the standard CCSIM start-up shell script. The destination directory for IWDB switch reports is specified by the value of the environmental variable IW_SIMULATED_DATA defined in the user's *.cshrc* file. The default is the current directory.

During the actual training session, the trainee accesses the switch reports generated by CCSIM as archived data using the normal IW procedures for such data. It is anticipated that training would involve some mixture of simulated archived data and real archived data. Simulated data is appropriate for training cases involving major network damage. Real data is appropriate for the many common cases of trunk and switch malfunctions that have symptoms that are not readily simulated.

Because CCSIM starts with an empty net, i.e., there is no traffic at time zero, switch reports do not represent realistic conditions until the simulation has run for some time. Consequently, CCSIM is programmed not to generate IWDB switch reports until the first hour of simulated time has elapsed. Also, it should be noted that simulated time and the switch report names do not advance to the next day at midnight. Lincoln Laboratory does not believe that this limitation is significant for TRAINER applications because traffic is normally low in the middle of the night, and the need for simulation runs spanning midnight is anticipated to be low.

A tape containing the latest version of CCSIM was shipped to DCEC on 13 September 1991.

4.2 OTHER EXTENSIONS AND IMPROVEMENTS

The implementation of Common Channel Signaling in CCSIM has been completed.

The LOAD-LEVEL command has been rewritten, and its speed has been significantly increased. This command is used to change traffic intensities in the network. One new use of the command is to model the behavior of the network over the course of a day rather than just at "busy hour." This is accomplished by using a series of LOAD-LEVEL commands in the command file. In this usage, the speed of execution of the command becomes important because many commands are issued in the course of a simulation run. To refine the traffic model further, local traffic is now affected by certain LOAD-LEVEL commands. A new variable has been added to the *<net>.reporting* file, which specifies the average variation around the normal local traffic at each switch. Nonreporting switches use a network-wide average value of 20 percent.

In the course of adding the TRAINER features described above, a previously observed problem with underscores in some variables in the *<net>.inval* file was corrected.

The CCSIM graphics interface, *ccsimtool*, was upgraded to work with Sun OpenWindows Release 2. The work involved coding changes to adjust fonts, colors, windows, and canvases. The appearance remains largely unchanged except that the trunk group (CLLI) window was reformatted and expanded to show new switch report fields now being collected in DSN-Europe. As a by-product, *nmestool*, the graphics interface for the old NMES, was also upgraded, but that program is no longer in active use.

Work done to verify and update the use of routines and variables in CCSIM resulted in a general cleanup and streamlining of the simulator. Forty-four obsolete routines were removed. Many unused or underused variables were removed. CCSIM now requires significantly less memory and is somewhat faster.

The statistical database (STATDB) utilities (see Section 2.5) were used to improve the fit between simulation results and real switch data. Real data were analyzed to obtain expected values and ranges for important CCSIM parameters. Then CCSIM output switch data were analyzed and compared to real data to verify that a European "busy hour" traffic matrix produced realistic switch reports. Using LOAD-LEVEL commands, CCSIM scripts were created to simulate normal 24-hr traffic patterns. This exercise demonstrated a useful CCSIM feature and validated traffic-dependent parameters.

CCSIM with its graphics program *ccsimtool* has been successfully ported to a Sun SPARCstation and installed at DCEC.

4.3 CCSIM DOCUMENTATION

"Using the Call-By-Call Simulator (CCSIM)" was issued on 16 November 1990. It includes instructions for building a network to be simulated by CCSIM.

The final version of the CCSIM Software Top-Level Design Document was issued on 19 September 1991 as a project memorandum. Changes since the draft version reviewed by DCEC in April 1991 respond to the points raised in that review.

Revision of the CCSIM Users Manual is under way. The revision will deal with the enhancements of CCSIM associated with the TRAINER application as well as recent changes in the graphics interface.

A draft document entitled "Common Channel Signaling as Implemented in CCSIM" was sent to DCEC for review.

4.4 PACIFIC NETWORK SIMULATION

A PAC91 network for CCSIM has been created using network descriptions and traffic files provided by DCEC files. The traffic files apparently include retried calls and will need modification to serve as the call intention required for CCSIM. The traffic intensities can be manually scaled to get reasonable grade-of-service results as was done with earlier Pacific networks, but this activity has not yet been undertaken. Other information is also needed to bring the PAC91 network up to the same level of match to the real network that was achieved for the European network at the end of FY90.

Further development of PAC91 involves analyzing data from the real Pacific network and experimenting with the CCSIM traffic and other files to get switch reports from CCSIM to acceptably approximate those from the real network. Work in this direction has involved contacting the Pacific (PAC) ACOC and obtaining NMSS configuration files and sample switch data. The data show there are up to 45 configured sites, with 15 switches reporting. Eleven sites that are named "E/O" represent collections of end offices; six sites have no name. Our current PAC91 network has 24 switches. Considerable work will be needed to bring the CCSIM network into acceptable agreement with the real network.

The NMSS XREF files showed 276 trunk groups connecting reporting switches with other network switches (both reporting and nonreporting). The trunk group sizes have a large range; many have 1 to 40 trunks and some have 50 to 162 trunks. In comparison, our current European CCSIM network has 36 sites with 120 trunk groups, and the trunk group sizes range only from 1 to 48.

Sample switch data show that the report format is essentially the same as in Europe, but the reporting interval is every 15 min as opposed to every 5 min.

5. TRAMCON ALARM INTERPRETER (TAI) EXPERT SYSTEM

A component of DCEC FY91 tasking for Lincoln Laboratory is DRTV-funded expert systems development for DCS transmission system control. This tasking is referred to as TRAMCON alarm integration, as in past reports. It had one major component in FY91, implementation of the TRAMCON Alarm Interpreter (TAI), and one minor component, maintenance and extension of the TRAMCON Event Generator (TEG).

5.1 TRAMCON SYSTEMS

The DCS has been upgraded in recent years to include digital microwave radios, multiplexers, and encryption devices with considerable self-monitoring and remote control capability. DCS sites in Europe primarily employing such transmission equipment form the Digital European Backbone (DEB). Those sites are themselves grouped into DEB segments of roughly 10 sites each with one TRAMCON system per segment. A few sites within each segment host TRAMCON master monitors. Each master monitor can run the whole segment's TRAMCON system, but only one at a time. Sites in the segment without a master monitor can be unmanned with their transmission equipment remotely monitored and controlled.

It is commonplace for a fault in a piece of transmission equipment to trigger a primary alarm in the faulted piece of equipment and a constellation of sympathetic alarms in the same piece of equipment, in directly connected pieces of equipment, and even in distant pieces of equipment to which the faulted one is only indirectly connected. The sympathetic alarms are a consequence of the inability of self-monitoring subsystems to distinguish some local failures, e.g., a demodulator failure in a multiplexer from remote failures, e.g., a modulator failure in the equivalent multiplexer at the opposite terminal of the bit stream they support. Thus, transmission equipment alarms can go off in equipment and in sites where no actual equipment fault has occurred.

It is impossible to distinguish all possible primary alarms from all possible sympathetic alarms using only the alarm pattern available at a single site. By reporting alarms simultaneously from every site in a segment, a TRAMCON system makes it possible for its operator to distinguish primary alarms from secondary ones. But that doesn't make it easy, for several reasons.

As presented by a TRAMCON system, all alarms appear equally valid; only the overall pattern of alarms reveals which alarms are sympathetic. The number of different patterns of alarms that can occur in a given segment is large. Transmission equipment failures are sufficiently uncommon that operators typically do not observe many different alarm patterns before the end of a tour of duty or before changes in the segment's equipment configuration change the patterns of alarms in the segment. TRAMCON operators have many other tasks besides the diagnosis of transmission equipment failures based on such alarm patterns.

5.2 TAI AND TEG

The TAI has been developed to explore the practicality of using an expert system to aid TRAMCON system operators in diagnosing transmission equipment failures from segment-wide patterns of equipment alarms. Implementation of a preliminary version of TAI was a major piece of the FY91 effort. It built on the FY90 work that developed the TEG, an expert system that generates the TRAMCON alarm patterns corresponding to user-selected transmission equipment faults. Maintenance of and modest extensions to TEG were also part of the FY91 effort.

TEG can be used to simulate a TRAMCON system as a source of alarms for TAI to interpret. The development of TAI has built on the work that went into developing TEG. TEG was designed from its conception as a knowledge-based system, representing in a declarative style knowledge of DEB transmission equipment, connections between pieces of equipment, and fault trees for different types of equipment. TAI is also a knowledge-based system making use of exactly the same declarative knowledge to interpret alarm patterns instead of generating them.

Both TEG and TAI use the expert system shell CLIPS (C Language Integrated Production System) developed by NASA/Johnson Space Center and furnished without charge to government agencies. Each is organized into three sets of files interpreted by CLIPS. One set consists of files of CLIPS "facts" that describe the types of transmission equipment found in DEB segments, the various ways that each type of equipment can fail, how lower level failures cause higher level failures, how both sorts of failures cause various equipment alarms, and which equipment alarms are monitored by TRAMCON systems. This set of files is used by both TEG and TAI to model the same types of transmission equipment.

A second set of files of CLIPS facts is used to describe a particular DEB segment. The description includes the sites in the segment, the specific pieces of the various types of transmission equipment found at each site, and the connections between pieces of equipment, both within and across sites. This set of files is used by both TEG and TAI to model the transmission equipment making up a particular DEB segment. Different sets are used to model the makeup of different segments. Note that a TRAMCON system requires an equivalent set of files to configure it for use in a specific DEB segment.

The third set of files includes CLIPS "rules" as well as facts. These rules and facts realize the particular program's algorithms and interface. These sets of files differ between TEG and TAI, producing their different behavior.

5.3 TEG AND TAI ALGORITHMS

TEG's main algorithm is quite straightforward. A user selects a specific type of failure to be simulated in a specific piece of transmission equipment. Knowledge of the mechanisms through which faults cause other faults in the same or different pieces of equipment is used to propagate faults throughout the segment. Knowledge of the specific pieces of equipment within a segment and their interconnections is also used in propagating faults. Finally, knowledge of which faults cause which alarms is used to generate the segment-wide alarm pattern.

TAI's algorithm is a little more complicated but exploits the same knowledge. It begins by inferring plausible higher-level transmission equipment faults from observed alarms. Knowledge of which higher-level equipment faults are caused by which lower-level faults is then used recursively to infer plausible low-level equipment faults. After this first stage of operation, each plausible fault could have caused at least one of the observed alarms.

A TEG-like procedure is then used to infer which alarms should have been observed if such a plausible low-level equipment fault actually occurred. If any of those alarms were not observed, that particular fault ceases to be plausible. After this second stage of operation, the surviving plausible alarms are those that could have caused, alone or in some specific combination, all the observed alarms and no others.

Often TAI will come up with several plausible low-level equipment faults as equally valid interpretations of an observed alarm pattern. All are usually in the same piece of equipment. The ambiguity in TAI's diagnoses is typically between closely-related subsystems of that one piece of equipment. These plausible faults produce the same alarms, both primary and sympathetic, as far as a TRAMCON system can observe.

5.4 TEG AND TAI INTERFACES

TEG actually devotes more facts and rules to the realization of its interface than to the realization of its algorithms. The result is a convenient, menu-driven interface that allows a user to simulate the remote control of transmission equipment or other TRAMCON system activities in addition to selecting a specific type of failure to simulate in a specific piece of transmission equipment. TAI devotes only a few facts and rules to the realization of its interface. Its interface is much simpler.

Ultimately, TAI is intended to read alarms directly from a TRAMCON master monitor. In lieu of that, the preliminary version of TAI reads them from TEG. After generating a set of alarms with TEG, a TEG user can command a description of the current alarms be written to a file in the form of CLIPS facts. The ability to write such a file is the primary FY91 extension to TEG. TAI simply loads these facts along with its other facts and rules at the beginning of a run and immediately begins interpreting a description of the transmission equipment faults that plausibly account for the set of alarms.

The preliminary version of TAI developed in FY91 has two alternative ways of describing those faults. One is used when TAI is run by itself. In that case, a brief description of each plausible fault is printed, annotated by a list of the specific alarms for which it could account. Those plausible faults, if any, that account for all the observed alarms are printed first. The plausible faults that could account for some of the alarms are printed next, in decreasing order of the number of alarms accounted for.

TAI can also be run in combination with the Network Simulation (NETSIM). NETSIM was originally developed in FY90 as part of an AFSC/RL-sponsored Lincoln Laboratory project called Expert Systems for Distributed DSC Control (EDC). During FY90, TEG and NETSIM were extended so that NETSIM could serve as a graphical interface to TEG for the selection of specific types of failures in specific pieces of transmission equipment and for the display of resulting alarms. Other aspects of TEG's interface are disabled when it is run with NETSIM. The graphical interface provides an intuitive and appealing showcase for TEG's main function, the simulation of TRAMCON system alarms.

The success in using NETSIM as a front end for TEG led to using NETSIM as a front end for TAI as well. Although it currently displays only those plausible faults that could account for all observed alarms, NETSIM has provided an intuitive and appealing showcase for TAI's diagnostic capabilities. That is particularly the case when the graphical interface is used, first, to select a specific piece of transmission equipment and fault; second, to display the resulting alarms produced by TEG with the associated sites and pieces of equipment highlighted; and third, to display the plausible faults produced by TAI, again with the associated sites and pieces of equipment.

5.5 TEG AND TAI FUTURE

During the last quarter of FY91, TEG and TAI with NETSIM as their user interface, were demonstrated to representatives of DISA/DCEC/DRTV, AFSC/RL/C3DA, and AFCC/CCSC/CO plus a senior TRAMCON system operator from USAFE. The potential utility to TRAMCON system operators of a system like TAI was apparent to all observers. Significant interest has been expressed in deploying a TAI-like adjunct to TRAMCON systems in a few years.

To demonstrate that the potential of TAI is even more compelling and to move towards the development of a field-deployable system, work is necessary in several areas. One area is interfaces; in part this means some evolution and perhaps specialization of the NETSIM front-end TEG and TAI. For example, more of TEG's functionality should be accessible through NETSIM than is now the case.

Work must also be done on interfacing with actual TRAMCON systems. A dynamic interface is required between TAI and a TRAMCON master monitor to obtain current alarms. Given the age of the TRAMCON systems and their software, that interface cannot require changes in existing TRAMCON hardware or software. It appears that TAI will have to mimic a TRAMCON display terminal with an operator at its keyboard to obtain alarm information while humoring the existing hardware and software.

Work is also necessary in the area of the declarative knowledge upon which TEG and TAI depend. The existing knowledge base has been superficially verified in the course of demonstrating TEG for experts in TRAMCON operation. In FY90, TEG was exercised by two CCSC staff recently returned to Tinker AFB from assignments in Europe as TRAMCON operators. They believed it produced correct results. More thorough testing is required before fielding a system. The knowledge base may also need to be extended to describe types of transmission equipment not currently known to TEG and TAI.

It is possible that such testing and extension will uncover gaps or errors in the knowledge base that require changes in how TEG and TAI model equipment, equipment failures, or how one failure causes another. The CLIPS facts that represent such knowledge in TEG and TAI are not themselves models but only collections of model parameter values. It is what information those facts store and how rules interpret that information that define the models. While modeling changes can have a pervasive impact on expert systems, the declarative manner in which TEG and TAI store most of the knowledge they exploit should minimize the work required to make any necessary modeling changes. The likelihood that extensive changes will be needed is small.

The existing knowledge base currently contains one set of files describing a DEB segment's sites, equipment, and connections. That description is not current; there are known to be some minor discrepancies between it and the current makeup of the segment. A fieldable TAI-like adjunct to a TRAMCON system will need to have available up-to-date descriptions of all segments, perhaps translated from equivalent TRAMCON system configuration files, or it will need to gather that information dynamically from its associated TRAMCON master monitor. Either a translation program or an appropriate dynamic interface must be developed.

Efficiency is the last area in which work is necessary. When large numbers of alarms are involved, it can take as much as 1 min per alarm for TAI to compute its list of plausible faults on a Sun 3/260. That is far too slow for field deployment. Simply using a new, faster Sun computer would help some. Rewriting

TAI's CLIPS rules could also produce improvements in efficiency. Changes in how TEG and TAI model equipment, equipment failures, or the causation of one failure by another could also lead to efficiency improvements. All must be tried to evaluate their impacts.

The main bottleneck in the current implementation of TAI appears to be the interpretive nature of CLIPS. It may be necessary to reimplement TAI and TEG in an algorithmic programming language such as C. The reimplemented programs would still be knowledge-based, still make use of declarative knowledge about equipment, connections between equipment, equipment failures, and so forth. Such programs could even be designed to read and parse the existing files of CLIPS facts that represent the knowledge. The program would simply implement the TEG and TAI algorithms and interfaces more directly and efficiently. Open questions remain: Is such a reimplementation necessary? When is it appropriate?

APPENDIX A
IWES SYSTEM DESIGN

The Integrated Workstation Expert System (IWES) is written in C and an expert system shell developed by the Artificial Intelligence Section of the Mission Planning and Analysis Division at NASA/ Johnson Space Center called CLIPS (C Language Integrated Production System). CLIPS, CLIPS updates, and CLIPS documentation are available at no cost for U.S. government work.

The IWES is the result of integrating an existing stand-alone expert system, the Network Management Expert System (NMES) into the Integrated Workstation (IW). As a result, some user options and internal code associated with the original system are obsolete in relation to the IW but still exist in IWES. Code still exists to communicate with the NMES graphics, the Call-by-Call Simulator (CCSIM), and a switch report translator. None of the stand-alone options or code are visible to the IW operator, but they will be visible to anyone involved with programming the IWES. In the future, all of the old NMES code that is not in use should be removed.

A.1 PROGRAM FLOW

The following sections describe major procedural routines that constitute IWES. These include the programming structures and the important functions executed in each major routine. A description of the CLIPS rules and C monitors is found in Appendix B.

In the following sections, variables and function names are set in Courier type, e.g., `nmes_reset()`, and file names are set in Courier Italic, e.g., *`iwes.c`*.Functions are identified by the "()" following them. Message names are underlined, e.g., polldone.

A.1.1 IWES Main Routine

The main procedure for the IWES is located in the file *`iwes.c`*. This file includes definitions of globally used constants and structures in *`nmes_struct.h`*, *`nmes_cnst.h`*, *`msgcnst.h`*, *`msgstruct.h`*, *`sck.h`*, and *`stat_utils.h`*. The main procedure calls routines to accomplish the following:

1. Parse the arguments – `parse_args()` (*`iwes.c`*)
2. Initialize IWES – `nmes_init()` (*`nmes_init.C`*)
   This routine
   - initializes CLIPS
   - loads the CLIPS rules
   - builds the IWES switch, link, and clli linked lists
   - opens a log file
   - reads in problem information
   - reads in control information
   - reads in combo related information.

3. Reset IWES – `nmes_reset()` (*nmes_reset.c*)
   This routine
   - resets CLIPS
   - asserts node facts into CLIPS
   - asserts clli facts into CLIPS
   - loads statistical database (STATDB) information
   - asserts IWES time facts into CLIPS
   - asserts auto-confirm facts
   - asserts problem suppression facts
   - resets switch and link information to zero.
4. Sets up the connection with the IWMD — `iwmdinit()` (*iwmd.c*)
5. Brings up the IWES UI window
6. Drops into a main loop that loops through
   - checking for input from IWES UI
   - checking for input from IWMD (*process_pipe_input.c* and *iwmd.c*)

IWES is in a checking-for-input mode until input is received from either the IWES UI or the IWMD. The following sections describe the program flow when specific messages are received from the IWMD.

A.1.2 Polldone Received from the IWMD

A polldone is received by the IWES through the `read_iwmd()` routine (*iwmd.c*). It is processed by `processiwmd()` (*iwmd.c*) which calls `process_switch_report_iwdb()` to access switch reports for the indicated time period. After the switch reports have been processed, `processiwmd()` calls `display_results()` to update the IWES UI.

`process_switch_report_iwdb()`, which is located in *process_switch_report.c*, calls the appropriate routines to accomplish the following:

1. Access the appropriate switch data by calling the IWDB routine `IWGetTrafficTable()`
2. Update the IWES time to the one in the new traffic table
3. Open a new log file if necessary
4. Execute a loop to do the following for each switch:
   - store switch data in the IWES network representation structures
   - assert to CLIPS a fact indicating that a switch report has been received
   - update the STATDB node buckets
   - run the switch monitors - Switch monitors detect anomalies in the switch report fields. For descriptions of switch monitors, see Appendix B, Section B.1.1 (Switch Monitors).
   - call `get_iwdb_clli()` to process trunk group information (*process_switch_report.c*)
5. Call `runes()` to check if it is time to run CLIPS (*runes.c*)

If IWES is being run with combo results, then CLIPS is run now. Otherwise it is run after a diagdone message is received. See Section A.1.4 (Running the Rule Portion of the Expert System).

get_iwdb_clli(), which is located in *process_switch_report.c*, calls the appropriate routines to accomplish the following:

1. Execute a loop to do the following for each trunk group:
   - store trunk group data in the IWES network representation structures
2. Run the clli monitors - clli monitors detect anomalies in the clli report fields. For descriptions of the clli monitors, see Appendix B, Section B.1.2 (Trunk Group Monitors).
3. Locate cllis without reports - See Appendix B, Section B.1.2.2 (Other Trunk Group Monitors).

A.1.3 Diagdone Received from the IWMD

A diagdone is received by the IWES via the read_iwmd() routine (*iwmd.c*). It is processed by processiwmd() (*iwmd.c*), which calls process_combo() if it is expecting information from the combo program to access the combo results for the indicated time period.

process_combo(), which is located in *process_combo.c*, calls the appropriate routines to accomplish the following:

1. Access the file combo result file pointer by calling the IWDB routine, IWGetIwnnReadDataStream()
2. Do the following for each combo result:
   - If a logic result, store the result in the switch structure to be used later
   - Else if a neural network result, call process_neuralnet() to process the result
3. Remove old text from switch state array
4. Remove old text from link state array
5. Call runes() to run CLIPS (*runes.c*). See Section A.1.4 - (running the rule portion of the expert system).

process_neuralnet(), which is located in *process_combo.c*, calls find_nn_switch_error() if the result is a neural network (NN) switch anomaly and nn_clli_error() if it is a neural network trunk group anomaly.

find_nn_switch_error(), which is located in *process_combo.c*, calls the appropriate routines to accomplish the following:

1. Find the switch in the linked list.
2. Create a history message about the anomaly.
3. Match the result with a known neural network switch anomaly.
4. Add the history message to the link state array.
5. Assert the anomaly into CLIPS facts.

nn_clli_error(), which is located in *process_combo.c*, calls the appropriate routines to accomplish the following:

1. Find the clli in the linked list structure.
2. Create a history message about the anomaly.
3. Match the result with a known NN trunk group anomaly.
4. Add the history message to the link state array.
5. Assert the anomaly into CLIPS facts.

A.1.4 Running the Rule Portion of the Expert System

runes() contains the calls to CLIPS that run the rules portion of the expert system. A detailed description of the rules and how they interact can be found in Appendix B, Section B.2.

The first thing runes(), which is located in *runes.c*, does is to check if it has received all the data it expects to receive before it runs CLIPS. If it has, it calls the appropriate routines to accomplish the following:

1. Run the interval monitors — interval monitors detect anomalies by analyzing fields in neighboring switches report fields. For descriptions of the interval monitors, see Appendix B, Section B.1.1.3 (Other Switch Monitors).
2. Assert the current time into CLIPS
3. Assert "retract facts no"
4. Run CLIPS to detect and monitor problems — run(-1) — see Appendix B, Section B.2 for descriptions of rules.
5. Assert "retract facts yes"
6. Run CLIPS to retract old facts — run(-1)
7. Call create_results_file() in (*results.c*) to create the IWES results file
8. Send recsdone messages to IWUI, RECUI, and MASKING

A.1.5 Creating the Results File

The IWES results file is created in the create_results_file() routine (*results.c*). create_results_file() first gets a file pointer by calling the IWDB routine, IWGetIwesWriteDataStream(). Next the timestamp is written to the file and then a loop is made through all the switches. If switch information is available, add_sw_info() is called to add the switch information to the file. Then add_tg_info() is called for each trunk group related to the switch.

add_sw_info(), which is located in *results.c*, calls the appropriate routines to accomplish the following:

1. Write the switch name to the file
2. Add the switch anomalies to the file
3. Add the switch history to the file
4. Call add_switch_logic_results() (*results.c*)
5. Call add_sw_prob() (*results.c*)

add_sw_logic_results(), which is located in *results.c*, calls the appropriate routines to accomplish the following for each experimental logic problem:

1. Write the problem to the file.
2. Write the type to the file.
3. Find the problem in the problem information linked list.
4. Write the problem information to the file.

add_sw_prob(), which is located in *results.c*, calls the appropriate routines to accomplish the following for each IWES problem:

1. Write the problem to the file.
2. Write the type to the file.
3. Find the problem in the problem information linked list.
4. Write the problem information to the file.
5. Add observations to the file.
6. Determine controls and add them to the file (if any should be recommended).

add_tg_info(), which is located in *results.c*, calls the appropriate routines to accomplish the following for each trunk group that has either an IWES problem or an experimental logic problem:

1. Write the trunk group number, source, destination, and clliname to the file
2. Add the trunk group anomalies to the file
3. Call add_tg_logic_results() (*results.c*)
4. For each problem on the trunk group, do the following:
   - write the problem to the file
   - write the type to the file
   - find the problem in the problem information linked list
   - write the problem information to the file
   - add observations to the file
   - determine controls and add them to the file (if any should be recommended).

5. Add the switch history to the file.

`add_tg_logic_results()`, which is located in *results.c*, calls the appropriate routines to accomplish the following for each experimental logic problem:

1. Write the problem to the file.
2. Write the type to the file.
3. Find the problem in the problem information linked list.
4. Write the problem information to the file.

A.1.6 Network Status Routines Called from CLIPS

There are six C routines called from CLIPS to update network status and problem arrays. These routines have to be declared and defined for CLIPS in a function called `usrfuncs()`, which is located in *iwes.c*. Other C functions called from CLIPS are also declared and defined there. For more information on defining C routines for use in CLIPS, please see the CLIPS documentation.

The following routines are called from CLIPS to update network status and problem arrays.

`iwes_switch_status()`, which is located in *iwes_status.c*, calls the appropriate routines to accomplish the following:

1. Compare the problem sent as a parameter from CLIPS with a list of expected problems to find a match.
2. Update the switch state array (history).

`iwes_clli_status()`, which is located in *iwes_status.c*, calls the appropriate routines to accomplish the following:

1. Compare the problem sent as a parameter from CLIPS with a list of expected problems to find a match.
2. Update the link state array (history).

`iwes_switch_problem()`, which is located in *iwes_problem.c*, calls the appropriate routines to accomplish the following:

1. Compare the problem sent as a parameter from CLIPS with a list of expected problems to find a match.
2. Update the problem array.

`iwes_clli_problem()`, which is located in *iwes_problem.c*, calls the appropriate routines to accomplish the following:

1. Compare the problem sent as a parameter from CLIPS with a list of expected problems to find a match.

2. Update the problem array.

nn_switch_status(), which is located in *nn_status.c*, calls the appropriate routines to accomplish the following:

1. Compare the problem sent as a parameter from CLIPS with a list of expected problems to find a match.
2. Update the switch state array (history).
3. Update the problem array.

nn_clli_status(), which is located in *nn_status.c*, calls the appropriate routines to accomplish the following:

1. Compare the problem sent as a parameter from CLIPS with a list of expected problems to find a match.
2. Update the link state array (history).
3. Update the problem array.

A.2 INPUTS, OUTPUTS, AND ASSOCIATED FILE FORMATS FOR IWES

A.2.1 Inputs to IWES

A.2.1.1 Switch Reports

Switch reports are read into the IWES, processed for specific interesting features that were abnormal for the time of day, which are then used in detecting network problem events and validating NN results. A call to IWDB routine `IWGetTraffic(timestamp)` is made in IWES routine process_switch_report_iwdb(), which is located in *process_switch_report.c*. This call returns a pointer to a structure that contains traffic information for each switch that responded to a poll for the time period specified.

A.2.1.2 Combo Results

Combo results are read into IWES every time interval. They are composed of both NN results and experimental logic results. The NN results are asserted into CLIPS, validated, and used to determine network problems and make recommendations for network management actions to be taken to correct the problems. The experimental logic results are stored in arrays. Later they are assigned coloring and actions and are written to the IWES results file along with the network problem events. A call to IWDB routine `IWGetIwnnReadDataStream(timestamp)` is made in IWES routine process_combo(), which is located in *process_combo.c*. This call returns a file pointer to the combo anomaly file. IWES uses this file pointer to read in anomalies line by line.

A.2.1.3 Switch Representation

Switch representation information is read into IWES upon initiation. IWES uses this information to set up a linked list of switch structures, which are used throughout all of the code. A call is made to IWDB routine IWGetSwitchTable() in IWES routine read_node_file(), which is located in *read_node.c*. This call returns a pointer to a structure that contains information about the switch.

A.2.1.4 Trunk Group Representation

Trunk representation information is read into IWES upon program initiation. Trunk information is added to the related IWES switch structures to be used throughout program execution. A call is made to IWDB routine IWGetNetworkTable() in IWES routine read_clli_file(), which is located in *read_clli.c*. This call returns a pointer to a structure that contains information about the trunk groups and their connectivity.

A.2.1.5 *Tokentext* File

The tokentext file contains a list of the anomalies that could be detected by the combo portion of the AI Utility. It indicates whether a problem is detected by the NN or the experimental logic program. It also indicates what type of element (switch or trunk group) the problem is found on and gives a short description of the problem. It is accessed directly from a file in the IWES routine readoutputnames(filename), which is located in *readoutputs.c*. The *tokentext* file is located in the *$IW_DATA/nn* directory.

A.2.1.6 Keyword List

The keyword list contains a list of words used as keys in the problem information file and the IWES results file. It is read into IWES upon program initiation. It is accessed directly from its file by the IWES routine read_keywords(), which is located in *prob_info.c*. The keyword file is located in the *$IW_DATA/aiu* directory and is named *keywords*.

A.2.1.7 Problem Information

Problem information, which consists of icon coloring, actions, and descriptions for each problem is read into IWES upon program initiation and when an updatecolors message is received from the Icon Coloring and Actions User Interface. It is accessed directly from its file in the IWES routine read_probinfo(), which is located in prob_info.c. The problem information file is located in the *$IW_DATA/aiu* directory and is named probinfo.

The format of the *probinfo* file is based on keywords. The list of acceptable keywords is obtained from the *keywords* file (see Section A.2.1.6). The first keyword must be "Problem:". IWES expects that all of the keywords following (and the text following them) are associated with the same problem until it reaches another "Problem:" keyword. The keywords and following text are interpreted as follows:

Problem:		
<problem_token>	Where:	problem_token must be one word that is either in the *tokentext* file or recognized by IWES as a network problem event

Coloring:		
<color>	Where:	color can be one of three words: hidden, yellow, or red
Action:		
<actions[i]>	Where:	action[i] = a line of text ending with a newline character
<keyword or EOF>		A keyword from the keyword list on a new line or an EOF (End Of File) indicates that all the actions have been read. The keyword case also indicates that another type of input follows.
Description:		
<description[i]>	Where:	Description[i] = a line of text ending with a newline character
<keyword or EOF>		A keyword from the keyword list on a new line or an EOF (End Of File) indicates that all the descriptions have been read. The keyword case also indicates that another type of input follows.

A.2.1.8 Preplans

Preplans allow a network management expert to tailor recommended actions and controls for specific switches and trunk groups. They are accessed directly from files in the IWES routine `get_sw_ppln_from_file()`, which is located in *pre_plans.c*. The preplan files are prepared manually using a text editor and are located in the *$IW_DATA/iwes_config/ppln* directory and have a filename format of:

preplan.X	where X is a three character site name e.g., *preplan.UXB*
or	
preplan.X.Y	where X is a three character site name and, Y is a short clli name e.g., *preplan.UXB.SVN095*

Preplan files are made up of a Header Line followed by one or more Problem Recommendations blocks, each starting with a problem name and ending with the word "End". The formats are as follows:

Header Line		
<sw_name or clli_name>	Where:	sw_name = name of the switch clli_name = name of the clli
Problem Recommendations		(repeat for all types of problems that have preplanned recommendations specified for the switch or clli)
<problem>	Where:	problem is one of a predetermined set of text strings associated with IWES problems. These text strings are hard-coded in IWES. They presently include:

		Switch_Outage Switch_Congestion (although no IWES recognition is currently available for this problem) Permanent_Seizure Signalling_Glare Signalling_Problem
Actions		
<actions[i]>	Where:	actions[i] = a line of text (80 characters maximum) ending with a newline character. The number of action lines must be less than 20.
Controls		This section is made up of pairs of lines for each recommended control. The number of such line pairs must be less than 75.
<control[i]>	Where:	control[i] = a line of text for the user interface to display (80 characters maximum) ending with a newline character.
<controlparameters[i]>		controlparameters[i] = a line of formatted text (80 chars max) ending with a newline character. This text is intended to be sent on to the control module. Presently, this feature is not complete.
End		

A.2.1.9 Control Format Information

Control information is accessed by the IWES in order to verify and format recommended controls accurately. It is accessed directly from files in IWES routines `read_sitetype_file`() and `read_syntax_file`(), which are located in *`read_ctrl.c`*. The control format information files are located in the *`${IW_DATA}/iwes_config`* directory and have file names and contents as follows:

`ctrl_types.sites`	- contains a list of the reporting sites followed by their site type
`ctrl_syntax.nr`	- contains a list of site types, actions, and controls followed by an associated syntax number
`ctrl_syntax_params`	- contains a list of syntax numbers followed by a set of parameter numbers
`ctrl_syntax_labels`	- contains a list of parameter numbers followed by a text label and a parameter definition.

IWES must go through four steps to get the complete set of control parameters. First it determines the site type, then the syntax number, then parameter numbers, and finally the actual parameters. Presently, only *`ctrl_types.sites`* and *`ctrl_syntax.nr`* are read into IWES.

A.2.1.10 STATDB Files

STATDB (stat) files are found in the *$DSNPATH/expert/stats/db* directory. They are created off-line using procedures described in Appendix C. They are read by IWES routine `load_stats()` (in *stats.c*) during initialization and stored in a hash-coded table for efficient reference by the statistical monitors.

Each stat file is an ASCII text file containing hourly statistics for a specific stat-type for a single switch or trunk group. The file name defines what data was used to compile the statistics for what network element (switch or trunk group) they apply and to what stat-type they correspond. The file name has the format:

"stat-" <date> "+" <additional_days> "-" <element> "." <stat-type>.

For example, the file *stat-900103+9-ezl.ccb* contains Call Condense Block (CCB) statistics for the switch EZL. The contents were computed from 10 days of archived data beginning 3 January 1990. The file *stat-900806+8-tjs052.povfl* contains percentage overflow data for trunk group number 052 at the TJS switch. In this case, the contents were computed from 9 days of archived data beginning 6 August 1990.

Each stat file record (line of text) has the following format:

<hour-of-the-day> <mean-value> <number-of-samples> <minimum> <maximum> <standard-deviation>

Because there are up to 24 hr of switch report data available, stat files normally contain 24 records. The original database (currently installed in Europe) uses integers for the hour: 0, 1, ... 23. These correspond to the beginning of the hour for which the statistics were computed. The newer database uses a mixed number for the hour: 0.5, 1.5, ... 23.5. This change enables the plotting utility to more accurately plot the data. The statistics are based on the switch reports time-stamped from the beginning of the hour to 55 min after the hour.

A.2.2 Outputs from IWES

A.2.2.1 Results File

The results file is written by IWES and read by MASKING and REC UI. A call to the IWDB routine `IWGetIwesWriteDataStream(timestamp)` is made in IWES routine `create_results_file()`, which is located in *results.c*. This call returns a file pointer to the IWES results file. IWES uses this pointer to write into the results file line by line. The results files are located in the *${IW_DATA}/YYMMDD/iwes* directory and have a file name format of:

YYMMDD.HHmm.iwes	Where:	YY = year
		MM = month
		DD = day
		HH = hour
		mm = minute
	e.g.,	*900927.1540.iwes*

The format of the results file is based on keywords. The list of acceptable keywords is obtained from the *keywords* file (see Section A.2.1.6). The first keyword in the results file is always "Time:" and the second is always "Switch:".

Until another "Switch:" keyword is written, all of the keywords and their text are associated with the last switch name written. Each switch can have three different types of information associated with it: general information associated with problems, entries for specific problems associated with the switch, and entries for trunk groups associated with the switch. The format of each of these is described below.

When a "Trunk_group:" keyword is written, all of the keywords and the text following it, are associated with the trunk group, until another "Trunk_group:" or "Switch:" keyword is written. Each trunk group can have two different types of information associated with it, general information associated with problems and entries for specific problems associated with the trunk group.

An entry is made for each switch for which problems were recognized by IWES either in the switch itself or on any of its trunk groups. If both switch and trunk group problems are present, the switch problems appear first in the file.

Header Line

Time:		
<Timestamp>	Where:	Timestamp is in the format YYMMDD.HHMM.

Switch Information

Switch:		
<sw_name>	Where:	sw_name = name of the switch.

General Information Associated with Problems at a Switch

- Either of the following may be excluded from the file if no information about them is present for this time period in IWES.

Anomalies:		
<Anomalies[i]>	Where:	Anomalies[i] = a line of text ending with a newline character. The next keyword or an EOF indicates the end of the anomaly text.
History:		
<History[i]>	Where:	History[i] = a line of text ending with a newline character. The next keyword or an EOF indicates the end of the history text.

Specific Problems Associated with the Switch

- There may be zero or more problems per switch. If there are no problems associated with the switch, then no problems will be entered, and the trunk group information will be written next. A "Problem:" keyword and information about the specific problem will be written for each problem at the switch.

All the information associated with the problem must be included before the next problem, switch or trunk group keyword is written.

Problem:		
<problem_token>	Where:	problem_token = a one-word token that is either in the *tokentext* file or is recognized by IWES as a network problem event

Information About Specific Problems Associated with the Switch

- All of the following keywords and their associated text need not be included for every problem.

Type:		
<problem_type>	Where:	problem_type = the type of problem. Either: "experimental_logic" or "iwes"
Coloring:		
<color>	Where:	color = one of three words: hidden, yellow, or red
Description:		
<Description[i]>	Where:	Description[i] = a line of text ending with a newline character. The next keyword or an EOF indicates the end of the description text.
Action:		
<Action[i]>	Where:	Action[i] = a line of text ending with a newline character. The next keyword or an EOF indicates the end of the action text.
Observations:		
<Observations[i]>	Where:	Observations[i] = a line of text ending with a newline character. The next keyword or an EOF indicates the end of the observations text.
Controls:		This section is made up of pairs of lines for each control recommended by IWES. The next keyword or an EOF indicates the end of the Controls section text.
<control[i]>	Where:	control[i] = a line of text for the user interface to display ending with a newlin' character
<controlparameters[i]>		controlparameters[i] = a line of formatted text ending with a newline character. This text will be sent on to the control module. Presently, formatting of the control parameters is not complete, and controlparameters are not being sent to the control module.

Trunk Group Information

Trunk_group:
<Clli_name> Num: <trunk_group_num> Src: <src> Dest: <dest>

	Where:	clli_name = name of the clli
		trunk_group_num = the trunk group number
		src = the source switch
		dest = the destination switch

General Information Associated with Problems on a Trunk Group

- Either of the following may be excluded from the file if no information about them is present for this time period in IWES.

Anomalies: <Anomalies[i]>	Where:	Anomalies[i] = a line of text ending with a newline character. The next keyword or an EOF indicates the end of the anomaly text.
History: <History[i]>	Where:	History[i] = a line of text ending with a newline character. The next keyword or an EOF indicates the end of the history text.

Specific Problems Associated with the Trunk Group

- There may be zero or more problems per trunk group. If there are no problems associated with the switch, then no problems will be entered, and the information for the next switch with problems will be written next. A "Problem:" keyword and information about the specific problem will be written for each problem on the trunk group. All the information associated with the problem must be included before the next problem, switch, or trunk group keyword is written.

Problem: <problem_token>	Where:	problem_token = a one-word token that is either in the *tokentext* file or is recognized by IWES as a network problem event.

Information About Specific Problems Associated with a Trunk Group

- All of the following keywords and their associated text need not be included for every problem.

Type: <problem_type>	Where:	problem_type = the type of problem. Either: "experimental_logic" or "iwes".
Coloring: <color>	Where:	color = one of three words: hidden, yellow, or red.

Description:		
<Description[i]>	Where:	Description. The next keyword or an EOF indicates the end of the description text.
Action:		
<Action[i]>	Where:	Action[i] = a line of text ending with a newline character. The next keyword or an EOF indicates the end of the action text.
Observations:		
<Observations[i]>	Where:	Observations[i] = a line of text ending with a newline character. The next keyword or an EOF indicates the end of the observations text.
Controls:		This section is made up of pairs of lines for each control recommended by IWES. The next keyword or an EOF indicates the end of the Controls section text.
<control[i]>	Where:	control[i] = a line of text for the user interface to display ending with a newline character.
<controlparameters[i]>		controlparameters[i] = a line of formatted text ending with a newline character. This text will be sent on to the control module. Presently, formatting of the control parameters is not complete, and controlparameters are not being sent to the control module.

A.2.2.2 Log File

The log file contains additional observations about network status. It is written by the IWES routine `write_log()`, which is located in *com_package.c*. The log is written into the *${IWDATA}/log/iwes* directory and has the file name format:

YYMMDD.iwes.log	Where:	YY = year
		MM = month
		DD = day
	e.g., *900927.iwes.log*	

The file is made up of entries with the following format:

<time> <observation>   Where: time is in the form HH:MM:SS observations = a line of text.

There are as many entries as observations during the day. A new file is started when a new day begins.

A.3 MESSAGE DISPATCHER MESSAGES

A.3.1 Messages Received by IWES

Messages from the IW message dispatcher are read in IWES routine `read_iwmd()`, which is called by *`processiwmd()`*. In `processiwmd()`, IWES analyzes the messages it has received and determines what actions it should take in response. Both `read_iwmd()` and `process_iwmd()` are located in *`iwmd.c`*. The following are the messages currently used by IWES.

A.3.1.1 polldone

Format: <src> <dest> polldone <timestamp>

When IWES receives a polldone message, it accesses raw switch reports from the IWDB, runs monitors to detect switch report anomalies, and asserts switch report anomalies into CLIPS as facts. It also runs statistics monitors that generate Alerts when values exceed thresholds.

A.3.1.2 diagdone

Format: <src> <dest> diagdone <timestamp>

When IWES receives a diagdone message it reads the NN output file and asserts NN anomalies into CLIPS as facts.

A.3.1.3 updatecolors

Format: <src> <dest> updatecolors

When IWES receives an updatecolors message it rereads the problem information file (*probinfo*) to get updated icon coloring and actions.

A.3.2 Message Sent by IWES

IWES sends only one message to the message dispatcher. It is sent in `processiwmd()`, which is located in *`iwmd.c`*.

A.3.2.1 recsdone

Format: iwes <dest> recsdone <timestamp>
  Where: dest = REC UI, IWUI, and MASKING.

IWES sends a recsdone message to the user interface and the masking program after it has run CLIPS and written out its results to the results file. Upon receiving the recsdone message, REC UI and MASKING access the IWES results file. Currently, IWUI receives the recsdone message but does not access the IWES results file.

APPENDIX B
IWES MONITOR AND RULE DESCRIPTIONS

This appendix contains descriptions of the monitors and the rules used in IWES. The expert system monitors are implemented as C routines and the rules are written in CLIPS. The C monitors perform the task of filtering through the data and detecting specific interesting features that were abnormal for the time of day in the raw data. The CLIPS rules combine these features to identify network problem events and to validate neural network anomalies. Details concerning the logic contained in the monitors are located in Section B.1 and details concerning the CLIPS rules are located in Section B.2 of this appendix.

During FY91 Lincoln Laboratory was requested to reinstate IWES's ability to recognize problems by using features detected by its monitors in addition to its FY90 capability of only validating anomalies detected by the neural network (NN). Most of the fixed-threshold (nonstatistical-database) monitors needed to do this were available in the FY90 system but the rules that use the monitor results to identify the problems had been removed from the rule base. As a result, the major difference between the description of the IWES monitors in our FY90 Annual Report, Appendix B, Section B.1, and the description to follow is the addition of new STATDB monitors. Because all of the rules described last year were related to validation of neural network anomalies, there are many differences between the rules described here (Section B.2) and those described in last year's report.

B.1 IWES MONITORS

Monitors are C language routines that process switch report data and/or the outputs of other monitors and store the results in a set of C language structures representing the network. Many of them also work to help maintain (in C arrays) lists of switches and trunk groups that have anomalous (nonnormal) states. The items on the lists combine the switch or trunk group with an anomaly identifier. When a monitor recognizes an anomaly, it appends an item to the appropriate list. It may also remove an item previously put on the list. There can be more than one item on the list for a particular switch or trunk group, but in such a case, the anomaly will be different. These lists are reported to the IW as "History."

Many of the monitors assert their findings as CLIPS facts. These facts are then used by CLIPS rules to identify network problem events and validate neural network anomalies.

The order in which individual monitors are run is important because some use the results generated by others. In the following sections, they are described in the order in which they run in IWES.

B.1.1 Switch Monitors

The monitors in this and the following section are run as each switch report arrives. Monitor names are those used in IWES. The program name is followed by an English language name intended to be descriptive of the function of the monitor.

B.1.1.1 Switch Report Register Monitors

These monitors check the individual registers in the switch report for a nonzero value that can indicate an abnormality with the system. The checks all use the following logic:

```
IF the value of the register is greater than zero
THEN
    assert for CLIPS the fact that switch X showed an instance of a nonzero register for this time cycle
    append a register-failure state to the switch state list
ELSE
    remove any register-failure state for switch X which might be left from a previous cycle
```

The following includes a list of the switch monitor routines and the individual registers they are checking. They are all found in the file *switch_monitor.c*.

Monitor - `find_no_mf_receiver_free`

Check the switch report RCVR MF Receiver Overflow register for evidence that attempts to assign a Multifrequency (MF) receiver found none free (overflow occurred).

Monitor - `find_no_dialing_receiver_free`

Check the switch report RCVR DGT Receiver Overflow register for evidence that attempts to assign a Digitone receiver found none free (overflow occurred).

Monitor - `find_cp_overflows`

Check Call Processing (CP) Registers for
- CCB overflow during attempt to seize a Call Condense Block (CCB)
- CP failures due to illegal software conditions (CP TRAP)
- CP failures due to unexpected results detected (CP SUIC)
- Call originations denied during warm and cold restarts (INITDENY).

Monitor - `find_mf_radr_overflows`

Check MF RADR Registers for
- MF failure to get response from receiver within the lower delay threshold (RADLDLYP)
- MF failure to get response from receiver within the upper delay threshold (RADUDLYP).

Monitor - `find_dial_radr_overflows`

Check Digitone RADR Registers for
- DGT failure to get response from receiver within the lower delay threshold (RADLDLYP)
- DGT failure to get response from receiver within the upper delay threshold (RADUDLYP).

Monitor - `find_cpu_overflows`

Check CPU registers for
- CPU mismatch interrupts due to interprocessor differences between the two CPUs (MTCHINIT).

CPU data store, program store, CMC link, or CMC data port system busied following a mismatch or trap interrupt (CPUFLT)

CPU warm restart and cold restart (SYSWINIT, SYSCINIT)

CPU simplex -

SYNCLOSS - Processor made simplex following mismatch interrupt

MSYLOSSU - Time usage of simplex mode due to manual intervention

SSYLOSSU - Time usage of simplex mode due to system action.

Monitor - `find_cmc_overflows`

Check CMC registers for calls to diagnostics on the CMC as a result of errors and faults on the registers CMC 0 and CMC 1.

Monitor - `find_trmtcm_overflows`

Check TRMTCM registers for

TCM partial digit calls occurred — i.e., one or more digits received but not enough for translation (TCMPDIL)

TCM permanent signal calls occurred — i.e., seizure with no ensuing digits (TCMPSIG)

TCM vacant code calls — i.e., translation occurred but no matching pattern is found (TCMVACT).

Monitor - `find_trmter_overflows`

Check TRMTER registers for

TER miscellaneous system failure calls (TERSYFL)

TER reorder calls that can be caused by mutilated digits, forced release, unexpected stops, excessive digits for translation, invalid start signals, invalid translations, and others (TERRODR).

Monitor - `find_trmtrs_overflows`

Check TRMTRS registers for

TRS emergency treatment 1 for deflected calls for CBK, DCC, ARC, or anything sent to EA1 (TRSEMR1)

TRS emergency treatment 2 for deflected calls for CBK, DCC, ARC, or anything sent to EA2 (TRSEMR2).

Monitor - `find_dcm_overflows`

Check DCM registers for

DCM referrals of a digital trunk for diagnostics (DCMCCTDG)

DCM digital trunk diagnostic failures (DCMCCTOP).

B.1.1.2 STATDB Monitors

Statistical database (STATDB) monitors compare current data to corresponding STATDB values to detect any switch report stat value that is outside its expected range for the specific time of day and switch. The STATDB values and related parameters (number of samples, minimum, maximum, and standard deviation) for each hour of the day are stored in files that are created off-line using procedures described in Appendix C. These files are read in during program initialization as described in Appendix A, Section A.2.1.10.

The following switch report values and calculations are monitored by `check_node_stats` (in *stat.c*):

MF calls
Digitone calls
MF usage
Digitone usage
MF holding time
Digitone holding time
CCBs
Calls/CCBs

The same logic is used to check all the values against interpolated normal statistics from the STATDB. The following logic is used to check for either "high" values or "low" values and to generate the corresponding "high" or "low" Alerts.

```
IF the difference between the actual time-smoothed value and the STATDB normal (plus or minus
a threshold check epsilon) is greater than the STATDB standard deviation (times a user-settable factor)
    THEN
        Add one to a count of how many times this type of statistic has been high/low
        Add one to a count of high/low Alerts
        Set the return result to high/low
        Add one to the number of Alerts at the node
        Add one to the total number of Alerts
        Add one to the Alerts during the hour
        IF this is the first Alert in the period
            THEN
                Add one to the number of Alert periods
        IF this is the first Alert for the time
            THEN
                Add one to the number of Alert times
        Write a message to the scrollable log window that switch X showed an instance of a high/
          low Alert of type Y
        Assert for CLIPS the fact that switch X showed an instance of a high/low Alert of type Y.
```

B.1.1.3 Other Switch Monitors

These switch monitors are independent of NN anomaly recognition; they check for receiving switch reports, not receiving switch reports, and evidence of activity at neighbor switches.

Monitor - `mark_switch_report_received` (in *`switch_monitor.c`*)

Remove any not-reporting state information for the switch. (The "reporting" state is not considered an anomaly and is not kept on the switch state list.)

```
IF a switch report is received from switch X
THEN
    remove any not-reporting state for switch X that might be left from a previous cycle.
```

Monitor - `remove_switch_monitors` (in *`switch_monitor.c`*)

```
Remove switch monitor states when node is not responding
IF a switch report is not received from switch X
THEN
        remove any reporting state for switch X that might be left from a previous cycle, including
           switch report field states.
```

Monitor - `find_nodes_not_responding` (in *`interval_monitors.c`*)

Find the nodes that haven't responded during the last time period

```
IF switch X is a reporting switch
AND switch X did not report this cycle
THEN
        assert for CLIPS the node-not-responding fact for switch X
        For each clli associated with the nodes:
           update link state for associated trunk groups
           assert for CLIPS the fact no-clli-information-received
ELSE
        remove any node-not-responding states for this switch that might be left from a previous
           cycle.
```

Monitor - `find_no_outgoing_attempts` (in *`interval_monitors.c`*)

Check for evidence of outgoing call attempts by switch X seen as incoming calls from switch X at its neighbors

```
IF one or more neighbor switches to switch X reported
AND the total incoming attempts from switch X seen by those neighbors was not greater than zero
THEN
        assert for CLIPS the fact that neighbors-see-no-incoming-signals from switch X at this time
        append a no-outgoing-attempts state for switch X to the switch state list
ELSE
        remove any no-outgoing-attempts state for this switch that might be left from a previous
           cycle.
```

B.1.2 Trunk Group Monitors

A normal switch report contains OM reports for each of the trunk groups that connect the switch to its neighbors. Each trunk group report is identified by a group number. Communication problems during switch polling may cause the loss of any or all of the trunk group reports associated with the poll. The following monitors are run as the switch report is being processed. They are run for all the trunk group reports that actually arrive. An additional monitor is run at the end to identify trunk groups for which no report was received.

B.1.2.1 Trunk Group Register Monitors

These monitors check the individual registers in the trunk group switch report for a nonzero value that can indicate an abnormality with the system. The monitors all use the following logic:

```
IF the value of the register is greater than zero
THEN
        assert for CLIPS the fact that trunk group X showed an instance of a nonzero register for
        this time cycle
        append a register-failure state to the trunk group state list
ELSE
        remove any register-failure state for trunk group X that might be left from a previous cycle.
```

The following includes a list of the trunk registers checked. They are all found in a single routine in the file *report_monitor.c*.

Monitor - `check_fail_clli_report`

Check the following trunk registers for
- Maintenance Busy Usage (MBU) on trunks due to manual actions
  - adjust `adjusted_inservice` variable for holding time calculations
- System Busy Usage (SBU) on trunks due to system problems
  - adjust `adjusted_inservice` variable for holding time calculations
- Incoming failure including permanent signals, partial digits, mutilated digits, receiver problems, etc. (INFAIL)
- Failure to effectively seize the chosen trunk due to hardware problems (OUTFAIL)
- Machine detected glare on the selected trunk (GLARE)
- Preemption attempts that were unsuccessful because there were no trunks available with a lower precedence level (PREOVFL).

B.1.2.2 Other Trunk Group Monitors

Monitor - `find_low_ht_calls` (in *report_monitors.c*)

Find trunks with many connections but low holding time

```
IF there have been outgoing attempts on the trunk group
AND the holding time is less than 20 s but more than 0 s
AND there are trunks in service
AND there are at least 36 calls per circuit per hour
THEN
        assert for CLIPS the fact that there is an instance of low-ht-with-calls for this time cycle on this trunk group
        append a low-ht-with-calls state to the trunk group state list
ELSE
        remove any low-ht-with-calls state that might be left from a previous cycle.
```

Monitor - `find_max_usage_few_calls` (in *`report_monitors.c`*)

Find trunks with high usage and few calls, i.e., more usage than the current time cycle calls would be expected to generate

```
IF there are fewer calls than there are trunks in service
AND the maximum usage the calls could generate is less than the usage register indicates
THEN
        assert for CLIPS that there is high_usage_few_calls for this time cycle
        append a high-usage-few-calls state to the trunk group state list
ELSE
        remove any high-usage-few-calls state that might be left from a previous cycle.
```

Monitor - `find_zero_usage_calls` (in *`report_monitors.c`*)

Check for calls with no usage

```
IF there is at least one outgoing or incoming call connection
AND the TRU usage register reports zero usage
AND there is at least one trunk in service after accounting for MBU and SBU
THEN
        assert for CLIPS the fact that there is an instance of zero_usage_calls for this time cycle on this trunk group
        append a zero-usage-with-calls state to the trunk group state list
ELSE
        remove any zero-usage-with-calls state that might be left from a previous cycle.
```

Monitor - `find_clli_trunks_down` (in *`report_monitors.c`*)

Check trunk group report for evidence that the number of trunks in service is less than the equipped value and/or is less than the number reported previously

```
IF the reported number of trunks in service is less than the reported equipped number
THEN
        append a decreased_capacity state for this trunk group to the trunk group state list
ELSE
        remove any decreased_capacity state for this trunk group that might be left from a previous cycle.
```

```
IF the reported number of trunks in service is less than the number reported previously
THEN
        assert for CLIPS the fact that a capacity change has occurred on this trunk group during this
           cycle.
```

Monitor - `find_hundred_overflow` (in *`report_monitors.c`*)

Check trunk group report for the case where the overflow peg count equals the outgoing attempts peg count and the usage value is zero. This situation corresponds to one type of trunk failure.

```
IF there are outgoing attempts on the trunk group
AND the overflows equal the outgoing attempts
AND the trunk usage is zero
THEN
        assert for CLIPS the fact that a hundred-percent-overflow-zero-usage condition exists
           on this trunk group at this report time.
```

Monitor - `find_zero_overflow` (in *`report_monitors.c`*)

Check trunk group report for the case where there are outgoing attempts, no overflows, and no usage. This situation can occur if 100 percent SKIP controls are put on at both ends of the trunk group.

```
IF there are outgoing attempts on the trunk group
AND the overflow peg count is zero
AND the trunk usage is zero
THEN
        assert for CLIPS the fact that a zero-percent-overflow-zero-usage condition exists on
           this trunk group at this report time.
```

The following monitor is run after the monitors described above have been run for all the trunk groups reported in a switch report.

Monitor - `find_cllis_without_reports` (in *`report_monitors.c`*)

By going through a list of trunk groups that exist at the reporting switch and for which reports are expected, find and mark those missing.

```
FOR all trunk groups in the C structure representing switch X
DO
IF the time stored in the trunk group representation does not equal the time of the current switch
      report
THEN
    assert for CLIPS the fact that a no-tg-information-received condition exists for this trunk
       group at this report time
    append a no-clli-report state for this trunk group to the trunk group state list
    remove all other trunk group states for this trunk group that can no longer be believed in the
       absence of a trunk group report
    ELSE
        remove any no-clli-report state for this trunk group that might be left from a previous cycle.
```

B.2 IWES RULES

IWES contains both rules for detection of network problem events and confirmation of NN anomalies. Section B.2.1 describes the rules for detection of network problem events, and Section B.2.2 covers the rules for confirmation of NN anomalies. Some additional rules for updating time facts, checking fact integrity, and retracting old facts exist, but are not described in this document (located in `rules.clp`).

The following descriptions of the IWES rules omit some details that relate to CLIPS syntax and other internal programming issues in order to simplify the presentation without losing information relative to the functions performed by the rules.

B.2.1 Rules for Detection of Network Problem Events

The rules for detecting network problem events are divided into two types. One is a set of specific rules that match combinations of monitor results to create problem facts for each problem. The other is a set of general rules that use these problem facts to recognize, monitor over time, relate and report network problem events. Section B.2.1.1 describes the general rules for detection of network problem events, and Section B.2.1.2 describes the specific rules for detection of network problems.

B.2.1.1 General Rules for Detection of Network Problem Events

General rules create network problem events by recognizing formatted problem facts, created by specific rules. Once a network problem is created, additional general rules update it over time and detect the removal of the problem. The general rules make use of CLIPS structures named "deftemplates". These deftemplates are described in Section B.2.1.1.1. The rules themselves are described in detail in Section B.2.1.1.2. The following list identifies the functions of the general rules and shows the names of the rules that carry out those functions.

1. Recognize a new network problem event.
   `(sw-problem-event, tg-problem-event)`
2. Recognize a network problem event over time.
   `(sw-problem-event-still, tg-problem-event-still)`
3. Recognize an unsupported network problem event and mark for removal.
   `(iwes-only-not-confirmed-sw-problem-event,`
   `iwes-only-not-confirmed-tg-problem-event)`
4. Recognize a network problem event that is now detected by the NN and mark for removal. This will become a confirmed neural network anomaly.
   `(iwes-only-to-confirmed-sw-problem-event,`
   `iwes-only-to-confirmed-tg-problem-event)`
5. Override less significant network problem events.
   `(override-less-significant-sw-problem-events,`
   `override-less-significant-tg-problem-events)`

6. Report a new network problem event.
   (`report-new-sw-problem-event, report-new-tg-problem-event`)
7. Suppress the reporting of a new network problem event.
   (`suppress-new-sw-problem-event, suppress-new-tg-problem-event`)
8. Report a previously unsuppressed network problem event.
   (`update-previously-unsuppressed-sw-problem-event,`
   `update-previously-unsuppressed-tg-problem-event`)
9. Report a previously suppressed network problem event.
   (`update-previously-suppressed-sw-problem-event,`
   `update-previously-suppressed-tg-problem-event`)
10. Suppress the reporting of a previously unsuppressed network problem event.
    (`suppress-previously-unsuppressed-sw-problem-event,`
    `suppress-previously-unsuppressed-tg-problem-event`)
11. Suppress the reporting of a previously suppressed network problem event.
    (`suppress-previously-suppressed-sw-problem-event,`
    `suppress-previously-suppressed-tg-problem-event`)
12. Remove an unsuppressed network problem event.
    (`remove-unsuppressed-sw-problem-event,`
    `remove-unsuppressed-tg-problem-event`)
13. Remove a suppressed network problem event.
    (`remove-suppressed-sw-problem-event,`
    `remove-suppressed-tg-problem-event`)

B.2.1.1.1 Deftemplates Used in General Rules

The general rules for switch and trunk group problems make use of CLIPS structures called "deftemplates," that are similar to frames and define a group of related fields in a pattern, to define network problem events and problem structures. Deftemplate facts allow the programmer to abstract the structure of a fact by assigning names to each field found within the fact. As a result, the rules are not constrained to match every field in a fact in a specified order. Single or multiple fields of deftemplates are modified using a modify command and deftemplates are removed using a retract command (deftemplates are located in *`problemdeft.clp`*).

deftemplate `sw-problem-event`: This deftemplate defines a switch network problem event.

(field name)	;name of the problem
(field switch)	;switch where problem occurred
(field numt)	;number of times the problem occurred
(field time)	;time problem was updated

(field report-status)	;report status - should this problem be reported
(field previous-report-status)	;previous report status - was this problem reported last period
(field action)	;action to be taken

deftemplate `tg-problem-event`: This deftemplate defines a trunk group network problem event.

(field name)	;name of the problem
(field src)	;source of the tg where problem occurred
(field dst)	;destination of the tg where problem occurred
(field clliname)	;clliname of the tg where problem occurred
(field numt)	;number of times the problem occurred
(field time)	;time problem was updated
(field report-status)	;report status — should this problem be reported
(field previous-report-status)	;previous report status — was this problem reported last period
(field action)	;action to be taken

deftemplate `problem`: This deftemplate defines a problem.

(field name)	;name of the problem
(field el-type)	;switch or trunk group
(multi-field overrides)	;problems that are overriden by this problem

B.2.1.1.2 Description of General Problem-Detection Rules

General rules for network problem events look for problems detected only by the expert system monitors. These rules are written to run with or without the NN.

There are two sets of general rules for detection of network problem event anomalies: one for switches and one for trunk groups. Below is a description of the template for both sets of rules. The general rules for switch problems are located in *general-sw.clp* and the general rules for trunk group problems are located in *general-tg.clp*. For both switches and trunk groups there are the following 13 rules as listed in Section B.2.1.1:

`problem-event`: This rule looks for a new problem-event fact from the expert system monitor without seeing a corresponding anomaly from the NN.

```
IF an "NMES" fact exists for problem event X for this time period
AND a corresponding NN fact for problem event X does not exist for this time period
AND problem event X did not exist last time period
THEN
    assert a problem-event template fact for problem event X for this time period
    call a C routine to flag network-problem-event to the IWES log.
```

`problem-event-still`: This rule looks for a problem-event fact from the expert system monitor over time without seeing a corresponding anomaly from the NN.

```
IF an "NMES" fact exists for problem event X for this time period
AND problem event X existed last time period
AND a corresponding NN fact for problem event X does not exist for this time period
THEN
        update the time and set the action to update-problem in the problem-event template fact
            for problem
        call a C routine to flag network-problem-event-still-exists to the IWES log.
```

`iwes-only-not-reconfirmed-problem-event`: This rule removes any problem-event templates for problems that are not supported for the present time. This is done by checking the current time of the confirmed statement — if it is this time, keep it. If old time, remove it.

```
IF problem event X existed last time period
AND an "NMES" fact does not exist for problem event X for this time period
THEN
        update the time and set the action to remove-problem in the problem-event template fact
            for problem.
```

`iwes-only-to-confirmed-problem-event`: This rule removes any problem-event templates for problems that have now been detected by the NN. These later become confirmed NN problems.

```
IF problem event X existed last time period
AND a corresponding NN fact for problem event X exists for this time period
THEN
        update the time and set the action to remove-problem in the problem-event template fact
            for the problem.
```

`override-less-significant-problem-events`: This rule suppresses a less significant problem-event when a more significant related problem-event overrides it.

```
IF problem event X exists for the present time period with either report-new-problem or
            update-problem as an action
AND problem event Y exists for the present time period with either report-new-problem or
            update-problem as an action and unsuppressed as report-status
AND a problem template exists for problem event X which contains problem event Y in its
            overrides list
THEN
            update the report-status to suppressed in the problem-event template for problem event Y
            call a C routine to flag network-problem-event-suppressed to the IWES log.
```

`report-new-problem-event`: This rule calls a C routine to add the problem event to the problem array when a new problem is reported. It modifies the action to be `none` so that no further action will be taken for this problem in this time period. It also modifies previous-report-status to be `unsuppressed`. This rule has a low salience so that it will be done after the other rules.

```
IF problem event X exists for the present time period with report-new-problem as an action and
   unsuppressed as a report-status
THEN
      update the action to none and the previous-report-status to unsuppressed in the problem-
         event template for problem event X
      call a C routine to add problem event X to a status array
      call a C routine to add problem event X to the problem array
      call a C routine to flag network-problem-event-reported to the IWES log.
```

`suppress-new-problem-event`: This rule marks the problem event as being `suppressed`. It reinitializes report status to be `unsuppressed` for the next time period. It also modifies the action to be `none` so that no further actions will be taken for this problem event in this time period. This rule has a low salience so that it will be done after the other rules.

```
IF problem event X exists for the present time period with report-new-problem as an action and
   suppressed as a report-status
THEN
      update the action to none, report-status to be unsuppressed and the previous-report-status
         to suppressed in the problem-event template for problem event X
      call a C routine to add problem event X to a status array
      call a C routine to flag new-network-problem-event-suppressed to the IWES log.
```

`update-previously-unsuppressed-problem-event`: This rule calls a routine to update the problem event in the problem array when an updated problem is reported. It also modifies the action to be none so that no further action will be taken for this problem event this time period. This rule has a low salience so that it will be done after the other rules.

```
IF problem event X exists for the present time period with update-problem as an action,
   unsuppressed as a previous-report-status, and unsuppressed as a report-status
THEN
      update the action to none in the problem-event template for problem event X
      call a C routine to add problem event X to a status array
      call a C routine to add problem event X to the problem array
      call a C routine to flag new-network-problem-event-updated to the IWES log.
```

`update-previously-suppressed-problem-event`: This rule calls a routine to update the problem event in the problem array when a previously suppressed updated problem is reported. It modifies the action to be `none` so that no further action will be taken for this problem this time period. It also sets the previous-report-status to `unsuppressed`. This rule has a low salience so that it will be done after the other rules.

```
IF problem event X exists for the present time period with update-problem as an action,
  suppressed as a previous-report-status, and unsuppressed as a report-status
THEN
    update the action to none and the previous-report-status to be unsuppressed in the
      problem-event template for problem event X
    call a C routine to add problem event X to a status array
    call a C routine to add problem event X to the problem array
    call a C routine to flag new-network-problem-event-updated to the IWES log.
```

`suppress-previously-unsuppressed-problem-event`: This rule calls a routine to remove the problem event from the problem array when a suppressed previously unsuppressed problem is reported. It modifies the action to be `none` so that no further action will be taken for this problem event this time period. It also modifies previous-report-status to be `suppressed` and report-status to be `unsuppressed`. This rule has a low salience so that it will be done after the other rules.

```
IF problem event X exists for the present time period with update-problem as an action,
unsuppressed as a previous-report-status and suppressed as a report-status
THEN
    update the action to none, the report-status to unsuppressed and the previous-report-
      status to be suppressed in the problem-event template for problem event X
    call a C routine to add problem event X to a status array
    call a C routine to remove problem event X from the problem array
    call a C routine to flag new-network-problem-event-updated-but-suppressed to the
      IWES log.
```

`suppress-previously-suppressed-problem-event`: This rule marks a previously suppressed problem-event as still being suppressed. It reinitializes report status to be unsuppressed for the next time period. It also modifies the action to be `none` so that no further action will be taken for this problem event in this time period. This rule has a low salience so that it will be done after the other rules.

```
IF problem event X exists for the present time period with update-problem as an action,
suppressed as a previous-report-status and suppressed as a report-status
THEN
    update the action to none and the report-status to unsuppressed in the problem-event
      template for problem event X
    call a C routine to add problem event X to a status array
    call a C routine to flag new-network-problem-event-updated-but-still-suppressed
      to the IWES log.
```

`remove-unsuppressed-problem-event`: This rule calls a routine to remove the problem event from the problem array when a remove-problem is reported. It also retracts the problem event because it no longer exists. This rule has a low salience so that it will be done after the other rules.

```
IF problem event X exists for the present time period with remove-problem as an action,
unsuppressed as a previous-report-status
THEN
    retract the problem-event template for problem event X
    call a C routine to remove problem event X from a status array
    call a C routine to remove problem event X from the problem array
    call a C routine to flag network-problem-event-removed to the IWES log.
```

`remove-suppressed-problem-event`: This rule retracts a suppressed problem event because it no longer exists. This rule has a low salience so that it will be done after the other rules.

```
IF problem event X exists for the present time period with remove-problem as an action,
suppressed as a previous-report-status
THEN
    retract the problem-event template for problem event X
    call a C routine to remove problem event X from a status array
    call a C routine to flag suppressed-network-problem-event-removed to the IWES log.
```

B.2.1.2 Specific Rules for Detection of Network Problem Events

A specific rule for a problem creates an "NMES" fact for the problem if it is supported by the raw data and comparisons with the STATDB normals. NMES facts are written in a specific format and are used in general rules to detect and track network problem events if corresponding NN problems do not exist. Basically, these rules combine a few conditions and create a fact that can be matched in the general rules.

Most of the rules for diagnosing the problems recognized by IWES were developed by the European network manager, Mr. Thomas, using the Experimental Logic system and translated into CLIPS rules by Lincoln Laboratory. The Experimental Logic program, which was designed and implemented by GTE, has proved to be an excellent knowledge engineering tool not only because it captures the actual rules that the network management expert uses to diagnose problems, but it also allows the expert to increase his knowledge and refine his rules through experimentation. This enables Lincoln Laboratory personnel to discuss more complex network problem events with the network management expert that the Experimental Logic program is unable to capture. Some of these more complex network problem events require deviations from statistically normal data, information from neighboring switches and trunk groups or recognition over time, such as, abnormally high call condense block counts and trunks being permanently seized with unsupporting peg counts.

Specific rules exist for the following switch and trunk group network problem events:

Switch Problems

miscellaneous others
serious switch trouble - miscellaneous others
permanent signals
central processor traps
call condense block overflows
match initialization
trunk group hit
high mf receiver overload
mf receiver overload
severe mf receiver overload
high digitone receiver overload
severe digitone receiver overload
span diagnostics
* abnormally high call condense block count

Trunk Group Problems

glare
outfailures
infailures
excessive trunks out of service
significant trunks out of service
significant problem - glare
* one or more trunks permanently held up with unsupporting peg counts
restricted trunks

* These rules were not developed using the Experimental Logic system.

B.2.1.2.1 Specific Rules for Switch-Related Problem Events

The following rules are located in *switch-problems.clp*.

miscellaneous others

```
IF a fact exists for this time period indicating that there was a nonzero count in the TRMTER
   Register TERSYFL
AND an "NMES" fact for misc_other_tersyfl has not been created for this time period
THEN
      create an "NMES" fact for misc_other_tersyfl for this time period.
```

serious switch trouble - miscellaneous others

```
IF a fact exists for this time period indicating that the count in the TRMTER Register TERSYFL
   was greater than 10
AND a fact exists for this time period indicating that there were MF receiver overflows
   OR a fact exists for this time period indicating that there were DGT receiver overflows
AND an "NMES" fact for serious_sw_trbl has not been created for this time period.
THEN
       create an "NMES" fact for serious_sw_trbl for this time period.
```

permanent signals

```
IF a fact exists for this time period indicating that the count in the permanent signals register was
   greater than 20
AND an "NMES" fact for perm_signal has not been created for this time period
THEN
       create an "NMES" fact for perm_signal for this time period.
```

central processor traps

```
IF a fact exists for this time period indicating that the count in the CP register CPTRAP was greater
   than 0
AND an "NMES" fact for cputrap has not been created for this time period
THEN
       create an "NMES" fact for cputrap for this time period.
```

call condense block overflows

```
IF a fact exists for this time period indicating that the count in the CP register CCBOVFL was
   greater than 0
AND an "NMES" fact for ccb_ovfl has not been created for this time period
THEN
       create an "NMES" fact for ccb_ovfl for this time period.
```

match initialization

```
IF a fact exists for this time period indicating that the count in the CPU register MTCHINIT was
   greater than 0
AND an "NMES" fact for mach_int has not been created for this time period
THEN
       create an "NMES" fact for mach_int for this time period.
```

```
trunk group hit

    IF a fact exists for this time period indicating that the count in the TRMTCM Register TCMSIG
       was greater than 5
    AND a fact exists for this time period indicating that there were MF receiver overflows
    AND a fact exists for this time period indicating that the count in the MF RADR Register
       RADLDLYP was greater than 0
    AND an "NMES" fact for tg hit has not been created for this time period
    THEN
           create an "NMES" fact for tghit for this time period.

high mf receiver overload

    IF a fact exists for this time period indicating that there were MF receiver overflows
    AND a fact exists for this time period indicating that the count in the MF RADR Register
       RADLDLYP was greater than 3
    AND an "NMES" fact for highmfrcvrol has not been created for this time period
    THEN
           create an "NMES" fact for highmfrcvrol for this time period.

mf receiver overload

    IF a fact exists for this time period indicating that there were MF receiver overflows
    AND an "NMES" fact for mf_rcvr_ol has not been created for this time period
    THEN
           create an "NMES" fact for mf_rcvr_ol for this time period.

severe mf receiver overload

    IF a fact exists for this time period indicating that there were MF receiver overflows
    AND a fact exists for this time period indicating that the count in the MF RADR Register
       RADLDLYP was greater than 0
    AND a fact exists for this time period indicating that the count in the MF RADR Register
       RADUDLYP was greater than 0
    AND an "NMES" fact for severe_mf_rcvr_ol has not been created for this time period
    THEN
           create an "NMES" fact for severe_mf_rcvr_ol for this time period.

digitone receiver overload

    IF a fact exists for this time period indicating that there were DGT receiver overflows
    AND an "NMES" fact for dgt_rcvr_ol has not been created for this time period
    THEN
           create an "NMES" fact for dgt_rcvr_ol for this time period.
```

high digitone receiver overload

IF a fact exists for this time period indicating that there were DGT receiver overflows
AND a fact exists for this time period indicating that the count in the DGT RADR Register RADLDLYP was greater than 3
AND an "NMES" fact for highdgtrcvrol has not been created for this time period
THEN
create an "NMES" fact for highdgtrcvrol for this time period.

severe digitone receiver overload

IF a fact exists for this time period indicating that there were DGT receiver overflows
AND a fact exists for this time period indicating that the count in the DGT RADR Register RADLDLYP was greater than 0
AND a fact exists for this time period indicating that the count in the DGT RADR Register RADUDLYP was greater than 0
AND an "NMES" fact for severe_dgt_rcv_ol has not been created for this time period
THEN
create an "NMES" fact for severe_dgt_rcv_ol for this time period.

span diagnostics

IF a fact exists for this time period indicating that the count in the DCM Register DCMCCTDG was greater than 6
AND an "NMES" fact for span_diags has not been created for this time period
THEN
create an "NMES" fact for span_diags for this time period.

abnormally high call condense block count

IF a fact exists for this time period indicating that the CCB count was statistically high for the time of day
AND an "NMES" fact for abnormal-HI-ccb5 has not been created for this time period
THEN
create an "NMES" fact for abnormal-HI-ccb5 for this time period.

B.2.1.2.2 Specific Rules for Trunk-Group-Related Problems

The following rules are located in *tg-problems.clp*.

glare

IF a fact exists for this time period indicating that the count in the TRK register GLARE was greater than 6
AND an "NMES" fact for glare has not been created for this time period
THEN
create an "NMES" fact for glare for this time period.

outfailures

```
IF a fact exists for this time period indicating that the count in the TRK register OUTFAIL was
   greater than 5
AND an "NMES" fact for outfail has not been created for this time period
THEN
      create an "NMES" fact for outfail for this time period.
```

infailures

```
IF a fact exists for this time period indicating that the count in the TRK register INFAIL was
   greater than 5
AND an "NMES" fact for infail has not been created for this time period
THEN
      create an "NMES" fact for infail for this time period.
```

excessive trunks out of service

```
IF a fact exists for this time period indicating that at least 50 percent of the trunks on the trunk
   group are out of service
AND an "NMES" fact for exc_trks_oos has not been created for this time period
THEN
      create an "NMES" fact for exc_trks_oos for this time period
```

significant trunks out of service

```
IF a fact exists for this time period indicating that at least 25 percent of the trunks on the trunk
   group are out of service
AND an "NMES" fact for significant_oos has not been created for this time period
THEN
      create an "NMES" fact for significant_oos for this time period.
```

significant problem - glare

```
IF a fact exists for this time period indicating that the count in the TRK register GLARE was
   greater than 5
AND a fact exists for this time period indicating that the trunk group holding time was less than
   0.6
AND an "NMES" fact for sig_prob_glr has not been created for this time period
THEN
      create an "NMES" fact for sig_prob_glr for this time period
```

one or more trunks permanently held up with unsupporting peg counts

```
IF facts exist for the last 3 time periods indicating that there was usage on a trunk group without
   supporting peg counts
AND an "NMES" fact for permanent_seizure has not been created for this time period
THEN
      create an "NMES" fact for permanent_seizure for this time period.
```

`restricted trunks`

```
IF a fact exists for this time period indicating that there was 100 percent overflow with zero usage
AND a fact exists for this time period indicating that all trunks on the trunk group were in service
AND an "NMES" fact for restricted_trks has not been created for this time period
THEN
    create an "NMES" fact for restricted_trks for this time period.
```

B.2.2 Rules for Anomaly Verification

The rules for confirming IW NN diagnosed anomalies are divided into two types: a set of general rules to confirm or unconfirm and to monitor NN anomalies over time, and a set of specific rules to create "IWES" confirmation facts to be used by the general rules. The general rules for confirmation of NN anomalies are described in Section B.2.2.1 and the specific rules in Section B.2.2.2.

B.2.2.1 General Rules for Anomaly Confirmation

The general rules for anomaly confirmation require both an NN fact and an "IWES" confirmation fact for a problem to be confirmed. Other functions of these rules include watching the problem over time and detecting the removal of the NN or "IWES" confirmation fact. The inputs, outputs, and rules themselves are described in Sections B.2.2.1.1, B.2.2.1.2, and B.2.2.1.3. The following list identifies the functions of the general rules and shows the names of the rules that carry out those functions.

1. Confirm first time anomaly.
   `(nn-sw-anomaly, nn-tg-anomaly)`
2. Confirm subsequent anomaly.
   `(nn-sw-anomaly-still, nn-tg-anomaly-still)`
3. Unsupported first time anomaly.
   `(nn-sw-anomaly-disagree, nn-tg-anomaly-disagree)`
4. Unsupported subsequent anomaly.
   `(nn-sw-anomaly-disagree-still, nn-tg-anomaly-disagree-still)`
5. Remove confirmed anomaly when no NN anomaly present.
   `(nn-nmes-not-reconfirmed-sw-anomaly,`
   `nn-nmes-not-reconfirmed-tg-anomaly)`
6. Remove confirmed anomaly when NN anomaly no longer supported.
   `(nn-nmes-to-nn-only-sw-anomaly, nn-nmes-to-nn-only-tg-anomaly)`
7. Remove unsupported anomaly when no NN anomaly present.
   `(nn-only-not-reconfirmed-sw-anomaly,`
   `nn-only-not-reconfirmed-tg-anomaly)`

8. Remove unsupported anomaly when NN anomaly is confirmed.
   (`nn-only-to-confirmed-sw-anomaly`,
   `nn-only-to-confirmed-tg-anomaly`)

B.2.2.1.1 Inputs to General Rules for Anomaly Confirmation

There are two classes of anomalies: switch anomalies and trunk group anomalies. To report on a switch anomaly, the switch name is needed along with the status message. To report on a trunk group anomaly, the clli name, name of the source switch, name of the destination switch, and the status message is required. The Source and Destination names for the trunk groups are found in the CLLI fact, which is asserted when the expert system starts execution.

The general rules require the following facts:

1. An NN diagnosis of an anomaly for this time period. The CLIPS fact will be prefaced by an NN with the anomaly information contained in it. That the NN fact is from the current reporting time is assured by requiring a match between the variable ticks-now and the current time fact.
2. A "confirmed" or "not-supported" fact indicates that this is not the first time the anomaly has appeared. These facts are asserted by the NN rules when they are fired. Absence of this fact means that this is the first time the anomaly has been confirmed or not-supported.
3. An IWES fact that has been created by specific rules or an automatic confirm fact that has been asserted at the beginning of run time. The automatic confirm fact is used to confirm NN anomalies where sufficient information is unavailable that would allow the creation of rules to confirm the anomaly. The text describing these problems that is displayed to the user indicates that not enough information was available to confirm these problems. This feature is intended to allow new NN anomalies to be introduced without having to make corresponding changes to IWES.
4. The NET-TIME fact is the hour-minutes-second translation used by the logging procedures.
5. The CURRENT-TIME fact is the current time represented in tenths of seconds.

B.2.2.1.2 Outputs of General Rules for Anomaly Confirmation

The actions taken when an anomaly is confirmed or not supported are:

1. Retract the confirmed or not supported fact if it exists.
2. Assert a fact to be used the next time period.
3. Call a C routine, either `nn_switch_status` or `nn_clli_status`, where the appropriate status messages are logged and flags set.
4. Call C routine `clips_write_log` to log the expert system action.

The actions taken when an anomaly no longer exists are:

1. Retract the confirmed or not supported fact.
2. Call C routine `nn_switch_status` or `nn_clli_status` to remove anomaly status information.

There are two built-in fields in the call to the C status routines that are intended to be used to return related data for the user messages. The first field is a count of the number of times the expert system has seen the same problem. The second field is used when an unidentified NN anomaly is found. The other NN anomalies return the variable "sw-data," which is never used by the C routine.

B.2.2.1.3 Descriptions of General Confirmation Rules

There are two sets of general rules for confirmation of NN anomalies: one for switches and one for trunk groups. The two sets follow the same template. The following is the description of that template.

`nn-anomaly`: This rule is designed to confirm a first-time anomaly.

```
IF an NN fact for anomaly X exists for this time period
AND anomaly X was not confirmed last time period
AND a corresponding "IWES" fact for anomaly X exists for this time period
        OR an automatic confirm fact exists for anomaly X
THEN
        Assert a fact stating a confirmed anomaly for over-time evaluation
        Call a C routine to flag confirmation to IW
        Log conclusion.
```

`nn-anomaly-still`: This rule confirms that a problem found in the previous time period still exists.

```
IF an NN fact for anomaly X exists for this time period
AND anomaly X was confirmed last time period
AND a corresponding "IWES" fact for anomaly X exists for this time period
        OR an automatic confirm fact exists for anomaly X
THEN
        Retract the confirmed problem fact from previous time period
        Assert a fact stating a confirmed anomaly for over-time evaluation
        Call a C routine to flag confirmation to IW
        Log conclusion.
```

`nn-anomaly-disagree`: This rule finds an NN anomaly that the data does not confirm.

```
IF an NN fact for anomaly X exists for this time period
AND anomaly X was not confirmed last time period
AND a corresponding "IWES" fact for anomaly X does not exist for this time period
AND an automatic confirm fact does not exist for anomaly X
THEN
        Assert a fact stating an unsupported NN anomaly was found
        Call a C routine to flag nonconfirmation to IW
        Log conclusion.
```

`nn-anomaly-disagree-still`: This rule finds that an unsupported anomaly found in the previous time period still exists and is still unsupported.

```
IF an NN fact for anomaly X exists for this time period
AND anomaly X was unsupported last time period
AND a corresponding "IWES" fact for anomaly X does not exist for this time period
AND an automatic confirm fact does not exist for anomaly X
THEN
        Retract the unsupported statement from previous time period
        Assert a fact stating an unsupported anomaly was found
        Call a C routine to flag nonconfirmation to IW
        Log conclusion.
```

`nn-nmes-not-reconfirmed-anomaly`: This rule finds a confirmed fact from the previous time period with no new NN statement and removes the confirmed fact.

```
IF anomaly X was confirmed last time period
AND an NN fact for anomaly X does not exist for this time period
THEN
        Retract the confirmed fact
        Call a C routine to flag that the anomaly has disappeared.
```

`nn-nmes-to-nn-only-anomaly`: This rule removes a confirmed fact from the previous time period when the new NN anomaly is not confirmed by the switch report.

```
IF anomaly X was confirmed last time period
AND an NN fact for anomaly X exists for this time period
AND a corresponding "IWES" fact for anomaly X does not exist for this time period
AND an automatic confirm fact does not exist for anomaly X
THEN
        Retract the confirmed fact
        Call a C routine to remove the confirmed conclusions.
```

`nn-only-not-reconfirmed-anomaly`: This rule removes an unsupported fact from the previous time period when no new NN fact exists.

```
IF anomaly X was unsupported last time period
AND an NN fact for anomaly X does not exist for this time period
THEN
        Retract the confirmed fact
        Call a C routine to flag that the anomaly has disappeared.
```

`nn-only-to-confirmed-anomaly`: This rule removes an unsupported fact from the previous time period when the new NN statement is confirmed by the switch report.

```
IF anomaly X was unsupported last time period
AND an NN fact for anomaly X exists for this time period
AND a corresponding "IWES" fact for anomaly X exists for this time period
        OR an automatic confirm fact exists for anomaly X
THEN
        Retract the confirmed fact
        Call a C routine to remove the unsupported conclusions.
```

B.2.2.2 Specific Rules for Anomaly Confirmation

For each NN anomaly there should be a specific rule to create an "IWES" confirmation fact. "IWES" confirmation facts are created when the IWES monitors detect interesting features in the raw data corresponding to a specific NN anomaly.

The rules require the following facts:

1. A switch report data fact (or facts) that describe nonnormal conditions in the network. These facts are asserted by expert system monitors using the switch reports and the STATDB.
2. The NET-TIME fact is the hour-minutes-second translation used by the logging procedures.
3. The CURRENT-TIME fact is the current time represented in tenths of seconds.

The action taken in each case is to assert an "IWES" fact for the anomaly in question.

The NN and IWES in some cases use different terms (spellings) for the same anomaly. The following descriptions use the notation "IWES name / Neural Net name". The file name where the rule can be found is shown in parentheses following the rule name.

`outage remote/outage_remote` (in *nn-switch-problems.clp*)

Machine failure or switch outage_remote is recognized by no incoming calls and low holding time, possibly followed by high overflow if trunks go out of service or are put out of service. The NN anomaly is confirmed if IWES sees that the switch did not report and the neighbors of the switch did not see any calls from the switch. If the node did report or the neighbors did see calls from the switch, then the NN anomaly is not confirmed. The data from the trunks connected to the switch with the outage should indicate the accessibility of the switch.

`permanent seizure/permanent_seizure` (in *nn-tg-problems.clp*)

Permanent seizure, or trunks permanently held up but not with real traffic, is recognized with full usage and high holding time. The pattern continues with no supporting peg counts. There must be more usage on the trunks than the incoming and outgoing calls would be expected to generate, and there must be fewer total calls than the number of in-service trunks. These rules were used to confirm an NN diagnosis of permanent seizure. The difficulty is recognizing when a partial set of the trunk group has been seized, thus allowing some calls to get through while blocking other calls. The number of calls, usage, number of in-service trunks, holding time, and percent usage are of interest to this anomaly.

`facility hit/tropofade` (in *`nn-tg-problems.clp`*)

Facility hit is recognized by a very high peg count and a very low holding time. This anomaly was called "tropofade" in the original NN, LARS. There is a monitor that detects more than three calls per circuit with a low holding time and at least one in-service trunk group. The number of calls, usage, number of in-service trunks, holding time, and percent usage are of interest to this anomaly along with the normal statistics for these fields.

`trunk signaling problem/signalling_prob(lem)` (in *`nn-tg-problems.clp`*)

Trunk signaling problems are identified by low holding times sometimes accompanied by higher peg counts. A significant portion of outgoing attempts fail or experience glare. The ratio of failure is the total of outfails and glare in relationship to the number of attempts. If this ratio indicates more than 50 percent of the attempts fail, then the anomaly is confirmed.

APPENDIX C
BUILDING THE IW STATISTICAL DATABASE

Building the Integrated Workstation (IW) STATDB (STATDB) is an off-line procedure involving the processing of archived switch report data via UNIX shellscripts and programs written in AWK, a programming language well-suited for tasks involving simple mechanical data manipulation. Actually, *gawk*, the GNU Project's implementation of the AWK programming language is used. It conforms to the definition and description of the language in "The AWK Programming Language," by Aho, Kernighan, and Weinberger, with the additional AWK features defined in the System V Release 4 version of UNIX and some GNU-specific extensions.

stats_db.awk is the basic program used to create STATDB files. It takes an "xref" file and a list of switch reports for one switch only and produces a set of stat files for that switch. The "xref" file provides information about which of the trunk groups in the switch report are interswitch trunks and are therefore to be tallied in the statistics. The program will produce eight stat files for the switch, and seven stat files for each trunk group processed. The output files are created in the current working directory. (See Appendix A, Section A.2.1.10 for the file format and naming conventions used.) For example, to create stat files for UXB based on archived data from September 1990, one would type the following:

```
gawk  -f  $DSNPATH/expert/stats/stats_db.awk \
      $DSNPATH/expert/stats/90aug.xref  $IW_DATA/9009??/switch/*.uxb
```

The file following the -f option specifies the gawk program. The second parameter is the "xref" file used by the program to identify the trunk groups to be analyzed, and the last parameter expands (through the UNIX wildcard conventions '??' and "*") into a list of all switch reports for UXB during the month of September 1990. The "\" is another UNIX convention allowing the long line to be split for readability.

stats_db.awk also supports an option to process one specific trunk group at the switch rather than all trunk groups defined for the switch in the xref file. For example:

```
gawk -f $DSNPATH/expert/stats/stats_db.awk TRUNK GROUP=57
      $DSNPATH/expert/stats/90aug.xref $IW_DATA/9009??/switch/*.tjs
```

will produce only the trunk group stat files for trunk group 57 at TJS.

Whenever the network configuration changes, a new "xref" file must be generated. The AWK program, *config2xref.awk*, in *$DSNPATH/stats/tools/* reads an IW config table, e.g., *$IW_DATA/net_config/network.tbl*, and generates text for a corresponding 'xref' file.

build.statdb is a shellscript that will sequentially process a list of switch names and switch reports and build a STATDB. The script calls *stats_db.awk* for each switch in the list. The default mode will process all switches for the specified dates, and then make a second pass through all the data filtering out all values outside three standard deviations from the average computed in the first pass. This process is time-consuming. For example, to build a database for the 14 switches of the European DSN based on switch reports from 5 days requires approximately 10 hr on a Sun 3 workstation.

Another AWK program, *netstats_db.awk*, reads the individual STATDB switch files and produces net-stat files named "stat-<date>-net.<switch_stat_type>". These files contain hourly totals for the entire network. This data is useful when comparing overall network traffic changes and verifying a new STATDB.

Unfortunately, building an adequate database is not a straightforward process. The goal is a set of statistics for a "normal" network, but the switch reports do not generally contain only normal data. Traffic may be abnormal at certain times, there may be faults or other malfunctions present in the switches and/or trunk groups, and the switch reports themselves may be corrupted by data errors in the transmission path from the reporting switch to DISA-Europe. The *stats_db.awk* program provides a number of options to help in dealing with abnormalities, but the person creating the database should inspect the results and adjust as needed. For example, it may be necessary to leave certain reports out of the set entirely because of a previously known or discovered problem. To help with this process and to aid in evaluating the database, two shellscripts are available. *graph.avg* displays the average stats for a switch or trunk group that are found in the *stats/db/stat* directory. *graph.com* computes stat values for a given date and item (switch or trunk group) and then displays both the averages from the database and the values for the given date. Both of these scripts make use of the *xgraph* utility.

To add a new statistic or modify a current one, it is necessary to change the *stats_db.awk* program. Although the process involves editing the program text, Lincoln Laboratory personnel believe that it can be successfully carried out by someone using programming-by-example techniques without requiring a deep understanding of the AWK language.

Future work in this area should involve conversion of the *stats_db.awk* program to C to gain speed and to overcome the current shell limitation of five days of data.

GLOSSARY

AFSC	Air Force Systems Command
AI	Artificial Intelligence
CCB	Call Condense Block
CCSC	Command Communications Systems Center
CCSIM	Call-by-Call Simulator
CLIPS	C Language Integrated Production System
CP	Call Processing
DCEC	Defense Communications Engineering Center, Reston, VA
DCS	Defense Communication System
DEB	Digital European Backbone, part of the U.S. Microwave System deployed in Europe
DIMSS	Defense Integrated Management Support System
DISA	Defense Information Systems Agency
DSN	Defense Switched Network
EDC	Expert Systems for Distributed DSC Control
GMT	Greenwich Mean Time
IW	Integrated Workstation
IWDB	Integrated Workstation Database
IWES	Integrated Workstation Expert System
IWUI	Integrated Workstation User Interface
MF	Multifrequency
NETSIM	Network Simulation
NMES	Network Management Expert System
NMSS	Network Management Support System
NN	Neural Network
OT&E	Operational Test and Evaluation
PBX	Private Branch Exchange
REC UI	Recommendations User Interface
STATDB	Statistical Database
STATDB UI	Statistical Database User Interface
TAI	TRAMCON Alarm Interpreter
TEG	TRAMCON Event Generator
TRAMCON	The Transmission Monitoring and Control System used with DEB Microwave System
UI	User Interface

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE 30 September 1991	3. REPORT TYPE AND DATES COVERED Annual Report, 1 October 1990 – 30 September 1991

4. TITLE AND SUBTITLE

Knowledge-Based System Analysis and Control Defense Switched Network Task Areas

5. FUNDING NUMBERS

C — F19628-90-C-0002
PE — 620702F

6. AUTHOR(S)

Harold M. Heggestad

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Lincoln Laboratory, MIT
P.O. Box 73
Lexington, MA 02173-9108

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

DCA Engineering Group
1860 Wiehle Avenue
Reston, VA 22090-5500

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

ESD-TR-91-247

11. SUPPLEMENTARY NOTES

None

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

A major activity during FY91 has been the design and implementation of a new Artificial Intelligence (AI) Utility architecture for the Integrated Workstation (IW) used by network management personnel at Defense Information Systems Agency-Europe (DISA-Europe). For the new utility, Lincoln has developed a revised IW Expert System (IWES) that no longer depends on the neural network component of the old architecture for problem detection. In addition, the new IWES has monitors that produce Alert messages when reported values from the network indicate significant departures from normal values recorded in a statistical database. The new architecture also allows expert site personnel to adjust the behavior of the system through new user interfaces.

The IW, as installed in September 1990 and used throughout FY91, was subjected to operational test and evaluation at DISA-Europe in September 1991. The system, including the IWES, met user criteria and was recommended for acceptance as operational. Subsequently the new AI architecture was installed and demonstrated for site personnel.

Substantial work has been done to enhance the value of the Lincoln Call-by-Call Simulator (CCSIM) as a trainer for IW operators. The CCSIM Top-Level Design Document was issued, and a draft document on the Common Channel Signaling implementation was sent to the Defense Communications Engineering Center for review. Preliminary simulations of the current Pacific Defense Switched Network were carried out.

A component of DCEC FY91 tasking for Lincoln Laboratory is DRTV-funded expert systems development for the Defense Communication System transmission system control. This tasking is referred to as TRAMCON alarm integration, as in past reports. It had one major component in FY91, implementation of the TRAMCON Alarm Interpreter, and one minor component, maintenance and extension of the TRAMCON Event Generator.

14. SUBJECT TERMS

expert systems	system control	Defense Switched Network (DSN)
communications control	network management	communication network simulation

15. NUMBER OF PAGES
82

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT Unclassified